Apple DOS 3.3

Mit ausführlichen Programmbeispielen

Ulrich Stiehl

Tips und Tricks

2. Auflage



Hüthig

Ulrich Stiehl · Apple DOS 3.3 · Tips und Tricks

Ulrich Stiehl

Apple DOS 3.3

Tips und Tricks

Mit ausführlichen Programmbeispielen

2., ergänzte Auflage

Dr. Alfred Hüthig Verlag Heidelberg

Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der Markierung® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegt.

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
„Begleitdiskette zu Apple DOS 3.3", DM 28,—
ISBN 3-7785-1033-9

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stiehl, Ulrich:
Apple DOS 3.3 : Tips u. Tricks ; mit ausführl.
Programmbeispielen / Ulrich Stiehl. — 2., erg. Aufl. —
Heidelberg : Hüthig, 1984.
ISBN 3-7785-1049-5

Printed in Germany
Satz, Druck und Bindung: Druckerei Bitsch KG, Birkenau

Vorwort

Dies ist die erste deutschsprachige Darstellung des Diskettenbetriebssystems DOS 3.3 für den Apple II/II Plus/IIe. Im Gegensatz zu dem bislang nur in Englisch erschienen, ausschließlich für Assembler-Programmierer gedachten Buch ,,Beneath Apple DOS" von D. Worth und P. Lechner, das das 48K DOS sozusagen bis zum letzten Byte seziert, aber infolge fehlender praktischer Beispiele im großen und ganzen nur für Systemprogrammierer von Nutzen ist, liegt das Schwergewicht des vorliegenden Buches auf der ANWENDUNG von DOS 3.3 sowohl für Applesoft- als auch Assembler-Programmierer.

Der erste Teil behandelt ausführlich die dem Applesoft-Programmierer zur Verfügung stehenden DOS-Befehle, wobei die Textfiles wegen ihrer großen Bedeutung und der vergleichsweise komplizierten Handhabung besonders eingehend dargestellt werden. Viele Tricks aus meiner langjährigen Programmiererfahrung werden hier zum erstenmal geschildert. Aber auch im zweiten Teil findet der reine Applesoft-Programmierer insbesondere in dem Kapitel ,,Vermischte Tips, Tricks und Patches" zahlreiche verwertbare Anregungen. Im übrigen ist der zweite Teil für reine Assembler-Programmierer gedacht. Neben einer detaillierten Beschreibung der RWTS enthält dieser Teil elf vollständige RWTS-Anwendungsprogramme, die wie beispielsweise der TSL-Maker, der File-Reader, der RAM-Disk-Driver und die Fastbrun-Routine Techniken enthüllen, die bislang noch niemals publiziert (und wahrscheinlich auch noch nicht entdeckt) wurden. Anderenfalls wäre es unverständlich, weshalb solche ansonsten professionellen Anwenderprogramme wie zum Beispiel Visicale 3.3 oder Applewriter IIe 3.3 eine ,,schiere Ewigkeit" für das Einlesen von Textfiles benötigen. Unter Anwendung der hier erstmals beschriebenen Techniken würde sich die Einlesedauer auf einen Bruchteil der sonst benötigten Zeit reduzieren. Bei allen geschilderten Verfahren und Methoden wurde auf DOS-Kompatibilität Wert gelegt, d.h. es werden gewissermaßen nur ,,Tricks ohne Finten" dargestellt.

Bei den Anwendungsbeispielen wurde darauf geachtet, daß sie gleichzeitig einen sinnvollen Zweck erfüllen. So wird z.B. die Fastbrun-Routine anhand eines Apple II Plus Emulators für den Apple IIe erklärt, so daß man mit Hilfe dieser Routine nunmehr auf

dem Apple IIe Programme laufen lassen kann, die aus verschiedenen Gründen ausschließlich auf dem Apple II Plus liefen.

In der vorliegenden zweiten Auflage, die bereits drei Monate nach der Erstauflage erforderlich war, sind an einigen Stellen Setzfehler ausgemerzt sowie bei einem Programm (RWTS-Muster, Seite 113 - 115) vier fehlende Programmzeilen ergänzt worden.

An dieser Stelle möchte ich den zahlreichen Lesern danken, die mich spontan – meist auf dem Umweg über die Auskunft – angerufen haben. Deshalb, wenn Sie Fragen zu dem DOS-Buch haben, hier meine Telefon-Nummern und Anschriften: tagsüber 06221/489-1 (Dr. Alfred Hüthig Verlag, 6900 Heidelberg, Im Weiher 10) und abends 06221/800154 (6900 Heidelberg-Ziegelhausen, Rainweg 78).
Eingedenk der Tatsache, daß sich ein Leser 48 Stunden lang vergeblich bemüht hatte, den RAMDISK-Driver (Seite 170ff.) zum Laufen zu bringen, sei darauf hingewiesen, daß alle Programme exakt wie beschrieben laufen, wenn entweder das unveränderte DOS 3.3 von der System-Master-Diskette oder im Falle von Diversi-DOS die Version 2-C verwendet wird. Von den neueren Versionen, insbesondere 4-C, muß abgeraten werden, da hier viele Systemadressen nicht mehr stimmen.

Übrigens liegt inzwischen die Begleitdiskette zum DOS-Buch vor. Ferner erschien in diesem Monat der erste Band von „Apple ProDOS für Aufsteiger" einschließlich gesondert erhältlicher Begleitdiskette. Da ich zur Zeit noch an dem Buch „Apple Assembler – Vom Anfänger zum Profi" arbeite, das für August/September geplant ist, wird der zweite Band des ProDOS-Buches nicht vor September/Oktober herauskommen.

Ulrich Stiehl

Heidelberg, im Juni 1984

Inhalt

Teil I: DOS für Anwender und Applesoft-Programmierer

In diesem Teil I werden die Grundlagen des DOS 3.3-Betriebssystems in leichtverständlicher Form für Benutzer von Anwenderprogrammen sowie Applesoft-Programmierer behandelt. Einige Fakten sind bewußt vereinfacht dargestellt, damit der Neuling nicht durch zu viele technische Details abgeschreckt wird. Allerdings setzt dieser Teil ab Kapitel 2 Grundkenntnisse von Applesoft voraus, da dieses Buch kein Applesoft-Lehrbuch darstellen kann.

1. DOS für Benutzer von Anwenderprogrammen

1.1. Betriebssystem

DOS ist die Abkürzung für Disk Operating System (= Diskettenbetriebssystem). Ein Diskettenbetriebssystem ist ein in der Regel in Maschinensprache geschriebenes Systemprogramm, das der Verwaltung von Files (= Dateien) auf der Diskette dient. Files können Programme, z.B. Basic- oder Maschinenprogramme, Speicherauszüge, z.B. Grafikbilder, sowie Dateien im engeren Sinne sein, z.B. Brieftexte, Adressen, Telefonverzeichnisse usw.
Für den Apple II Plus und den neueren Apple IIe gibt es verschiedene Diskettenbetriebssysteme, nämlich

a) das Pascal-Betriebssystem
b) das ProDOS-Betriebssystem
c) das DOS 3.3-Betriebssystem

Betriebssysteme sind zumeist mehr oder weniger eng mit Programmiersprachen verquickt. So eignet sich z.B. das Pascal-Betriebssystem grundsätzlich nur zur Anwendung in Verbindung mit der Programmiersprache Pascal. ProDOS und DOS 3.3 sind für die Programmiersprachen Basic und Assembler (6502 Maschinensprache) gedacht. DOS 3.3 kann sowohl unter Applesoft Basic als auch unter Integer Basic eingesetzt werden, während ProDOS auf Applesoft Basic beschränkt ist.

1.1.1. Wann soll ich ProDOS und wann DOS 3.3 einsetzen?

ProDOS ist für externe Massenspeicher, d.h. Diskettenlaufwerke mit großer Kapazität sowie für Fest- und Wechselplattenlaufwerke ausgelegt. Damit eignet sich Pro-

DOS besonders gut für die Verwaltung von großen Datenmassen, z.B. Adreßdateien mit 10.000 Adressen usw. Der Preis für diese größere, externe Speicherkapazität ist ein um ca. 12.000 Speicherstellen geringerer interner Speicherraum, weil ProDOS als umfangreicheres Betriebssystem entsprechend mehr Platz beansprucht.
DOS 3.3 eignet sich mehr für kleinere externe Datenspeicher, z.B. die klassischen Apple-Diskettenlaufwerke mit ca. 143.000 Zeichen externer Speicherkapazität sowie die neueren von Fremdfirmen angebotenen Laufwerke (z.B. Cumara), die eine Speicherkapazität von mehr als 300.000 Zeichen haben.

1.1.2. Ist DOS gleich DOS?

Im Laufe der Zeit sind von Apple mehrere DOS-Verbesserungen erschienen, nämlich DOS 3, DOS 3.1, DOS 3.2, DOS 3.2.1 und schließlich DOS 3.3. In diesem Buch befassen wir uns ausschließlich mit DOS 3.3, da die vorangehenden Versionen, z.B. DOS 3.2 vom 16.2.1979, heute praktisch nicht mehr benutzt werden.

Aber auch DOS 3.3 ist nicht gleich DOS 3.3, denn im Laufe der Zeit sind von Apple-unabhängigen Firmen verschiedene DOS 3.3-Varianten veröffentlicht worden, z.B.

PRONTO-DOS
DAVID-DOS
HYPER-DOS
SUPER-DOS
DIVERSI-DOS
usw.

> **Hinweis:**
> Wenn in diesem Buch von Diversi-DOS die Rede ist, dann ist stets Version 2-C gemeint. Die neueren Versionen, insbesondere 4-C, sind nicht zu empfehlen, da bei diesen die Systemadressen nicht mehr stimmen.

Die beste DOS 3.3-Variante ist zweifellos Diversi-DOS, da dieses Betriebssystem bei einer Verbesserung der Zugriffsgeschwindigkeit um den Faktor 3-5 bei allen Datei-Arten einschließlich der sog. Textfiles keine Systemadreßänderungen vornimmt, so daß es mit DOS 3.3 völlig kompatibel ist, sofern der Programmierer im Falle eines Maschinenprogramms die Standard-Systemadressen benutzt und nicht wahllos im DOS „herumpokt". Diversi-DOS ist etwa so schnell wie ProDOS. Die anderen DOS 3.3-Varianten stellen in der Regel nur mäßige Verbesserungen dar, z.B. zumeist ausschließlich im Hinblick auf die Geschwindigkeit des Einladens von sog. Binärfiles (Basic-Programme, Maschinenprogramme, Grafik-Bilder).

Nachfolgend werden wir DOS 3.3 kurz als DOS bezeichnen. Alle Beschreibungen der Befehle und technischen Einzelheiten von DOS gelten auch für Diversi-DOS.

1.2. Laufwerk und Controller

Die klassischen Apple-Laufwerke sind Disk-Drives der Firma Shugart, die als weitgehend wartungsfrei und sehr zuverlässig bezeichnet werden können. Die Laufwerke sind durch ein Flachbandkabel mit einer sog. Controller-Karte verbunden, die in eine der sog. Slots („Schlitze") oder Steckplätze der Apple-Platine gesteckt werden können. Bei Apple II Plus können die Steckplätze 1-6 benutzt werden, während beim Apple IIe zusätzlich Slot 7 als möglicher Steckplatz in Frage kommt. Normalerweise sollte man die Controller-Karte in Slot 6 stecken, da zahlreiche Programme den Controller in Slot 6 erwarten. Ein einziger Controller kann mit zwei Laufwerken gleichzeitig verbunden werden, die dann entsprechend als Slot 6, Drive 1 bzw. Slot 6, Drive 2 angesprochen werden. Insgesamt können soviele Doppellaufwerke an den Apple angeschlossen werden, wie freie Steckplätze zur Verfügung stehen.

Die Apple-Laufwerke lesen eine Diskette stets von unten, d.h. der Lesekopf befindet sich sozusagen unterhalb der in das Laufwerk eingeschobenen Diskette. Man beachte, daß aus Gründen der Netzteilüberlastung zu einem bestimmten Zeitpunkt immer nur ein einziges Laufwerk eingeschaltet sein kann (erkenntlich an der rot aufleuchtenden „in use"-Lampe).

Die Pflege der Laufwerke beschränkt sich auf zweierlei:

1. kann man von Zeit zu Zeit— etwa zweimal pro Jahr— die Leseköpfe mit einer sog. Reinigungsdiskette reinigen. Hersteller von Reinigungsdisketten empfehlen eine häufigere, teilweise sogar wöchentliche Reinigung, doch scheint dies nach meinen Erfahrungen bei den Apple-Laufwerken nicht erforderlich zu sein.

2. kann man von Zeit zu Zeit die Laufwerksgeschwindigkeit neu justieren. Hierzu ist erforderlich, daß man das Gehäuse durch Entfernen der 4 Bodenschrauben abzieht und die hinten, unten rechts befindliche kleine Schraube („Potentiometer") nach rechts oder nach links dreht, je nachdem, ob die Rotationsgeschwindigkeit zu hoch oder zu niedrig ist. Die im Handel käuflichen Kopierprogramme enthalten meist eine entsprechende „Speed Adjustment"-Routine, die man zur Geschwindigkeitsregulierung laufen lassen muß.

1.3. Diskette

Disketten sind Datenträger mit direktem Zugriff, d.h. der Lesekopf kann per Programm jeden beliebigen Sektor der Diskette direkt lesen oder beschreiben. Demge-

genüber muß z.B. eine Datenkassette als Datenträger mit indirektem Zugriff solange vorwärts (oder sinngemäß rückwärts) gespult werden, bis der Lesekopf an der gewünschten Stelle angelangt ist.
Disketten können einseitig oder beidseitig mit einfacher oder doppelter Schreibdichte beschrieben werden. Für Apple-Laufwerke genügen einseitige Disketten mit einfacher Schreibdichte.
Ringverstärkte Disketten sind von Vorteil, doch darf der Ring bei Apple-Laufwerken nur ganz flach, d.h. wenig auftragend, sein. Eine optimale Zentrierung der Diskette erreicht man dadurch, daß man die Laufwerkklappe exakt in dem Moment behutsam schließt, in dem das rote „in use"-Lämpchen aufleuchtet.
Der Leseschlitz der Diskette, der sich — wie bereits erwähnt — unten befindet, sollte niemals mit (fettigen) Fingern in Berührung kommen. Ferner sollte man Disketten nicht biegen (und erst recht nicht knicken) und nicht auf den Monitor legen (wegen der Gefahr des Magnetfeldes). Schließlich sollte man Disketten keinen Temperaturschwankungen aussetzen. So kann z.B. das Verschicken von Disketten im Winter die Folge haben, daß der sich im Innern der Diskette befindliche Filz klamm wird und deshalb die magnetische Kunststoffscheibe (= die eigentliche Diskette) nicht mehr richtig rotiert.

1.3.1. Reset

Drücken Sie niemals die Reset-Taste, während ein Laufwerk eingeschaltet ist, denn wegen der damit verbundenen Stromzufuhr-Unterbrechnung wird in der Regel derjenige Sektor zerstört, auf dem der Lesekopf gerade ruhte, und zwar stets dann, wenn eine Schreiboperation stattfand. Bei Leseoperationen geschieht meistens nichts, doch wer weiß schon genau, ob gerade gelesen oder geschrieben wird? Deshalb Finger weg von der Reset-Taste, solange das „in-use"-Lämpchen an ist! Im äußersten Notfall Laufwerkklappe öffnen, Diskette halb oder ganz herausziehen und erst dann Reset drücken!

1.4. Speicherkapazität

Eine Diskette ist in Tracks (= konzentrische Spuren) und jede Spur ihrerseits in Sektoren eingeteilt. Die DOS-Diskette enthält 35 Spuren mit je 16 Sektoren. In einem einzelnen Sektor kann man 256 Zeichen (oder Bytes) speichern. Auf eine DOS-Diskette passen damit 35 mal 16 mal 256 = 143.360 Bytes (oder Zeichen). Eine Diskette umfaßt insgesamt 35 mal 16 = 560 Sektoren. Ein einzelne Spur enthält 16 mal

256 = 4096 Bytes oder 4 Kilobytes (1 Kilobyte = 1024 Zeichen), womit eine Diskette 35 mal 4 = 140 Kilobytes beinhaltet.

Die 35 Spuren sind von 0 bis 34 (oder hexadezimal von $00-$22) numeriert. Die Sektoren sind von 0-15 (oder hexadezimal von $00-$0F) numeriert. Die Spur 0 ist die äußerste und die Spur 34 die innerste Spur. Zwischen äußerster und innerster Spur beträgt der Abstand nicht einmal 1 cm, so daß die auf der Diskette genutzte Lesezone sehr schmal ist.

Auf den Spuren 0-2 befindet sich normalerweise das Betriebssystem, und auf der Spur 17 (hexadezimal $11) befindet sich der sog. Catalog, d.h. das Disketteninhaltsverzeichnis. Damit stehen auf einer normalen Diskette nur 35 minus 4 = 31 Spuren für reine Datenspeicherung zur Verfügung. Die Nettokapazität beträgt damit 31 mal 4096 = 126.976 Zeichen oder 31 mal 4 = 124 Kilobytes. Es besteht die Möglichkeit, DOS-lose Datendisketten zu erstellen, bei denen die Spuren 1 und 2 für Daten genutzt werden können, so daß dann die Nettokapazität 33 mal 4096 = 135.168 Zeichen beträgt.

1.5. Booten

„Bootstrap" bedeutet auf englisch eigentlich Stiefelschlaufe und „by one's own bootstrap" besagt soviel wie, daß man sich wie Münchhausen am eigenen Schopfe herauszieht. Ein Bootstrap-Programm oder kurz Boot-Programm ist ein sog. Urlader, der in aufeinanderfolgenden Stufen (ähnlich wie beim Raketenstart) das Betriebssystem einlädt. Das Boot-Programm befindet sich als nur 256 Bytes langes Programm in der Controller-Karte und wird im Falle von Slot 6 durch folgende Befehle aktiviert:

a) von Applesoft (Cursor Ü) aus durch PR#6 (bei Slot 5 PR#5 usw.)
b) vom Monitor (Cursor *) aus durch 6 Ctrl-P (bei Slot 5 5 Ctrl-P usw.)

Dies gilt für den Fall, daß der Apple bereits eingeschaltet war. Das reine Einschalten des Apple bewirkt automatisch den Boot-Vorgang (durch ein entsprechendes Monitorprogramm, das bei $FAA6 beginnt und dann nach $C600 im Falle von Slot 6 springt). Ferner bewirkt beim eingeschalteten Apple IIe die Tastenkombination Ctrl-Offener Apfel-Reset den Start des Boot-Programms.

Heutzutage gibt es kaum noch Apple II Plus Geräte mit weniger als 48K Speicherkapazität, und Apple IIe Geräte haben ohnehin stets 64K Speicherkapazität, so daß so-

genannte Master-Disketten, die das DOS in den jeweils höchstmöglichen Speicherbereich verschoben hatten, keine Bedeutung mehr haben.
Nach Beendigung des Boot-Vorgangs befindet sich das DOS in dem Bereich $9600-$BFFF (oder dezimal 38400-49151). Der für Programm und Daten freie Arbeitsspeicher beträgt damit ca. 36.000 Zeichen. Die Speicherverteilung sieht genauer gesagt folgendermaßen aus (technische Details siehe Teil II dieses Buches):

1.	$0000-$00FF:	Zero-Page oder Nullseite
2.	$0100-$01FF:	Stack oder Prozessor-Stapel
3.	$0200-$02FF:	Tastatureingabe-Puffer
4.	$0300-$03CF:	frei für kurze Maschinenprogramme
5.	$03D0-$03FF:	DOS-Vektoren
6.	$0400-$07FF:	Bildschirmspeicher
7.	$0800-$95FF:	frei für Programm und Daten
8.	$9600-$9CFF:	3 DOS-Datenpuffer
9.	$9D00-$AAC8:	DOS-Befehlsinterpreter
10.	$AAC9-$B5FF:	DOS File Manager (Entry bei $ AAFD)
11.	$B600-$BFFF:	DOS RWTS (Entry bei $ BD00)

Wenn das DOS in die Language Card (= die oberen 16K RAM) des Apple IIe oder Apple II Plus durch einen sog. DOS-Mover geschoben wird, sieht die Speicherverteilung in der Regel so aus:

1.	$0800-$BEFF:	frei für Programm und Daten
2.	$BF00-$BFFF:	DOS-Driver für Bank-Selecting
3.	$D000-$DFFF:	Bank 1 der Language Card frei
4.	$D000-$FFFF:	Bank 2 der Language Card DOS

Die DOS RWTS würde dann z.B. bei $F600 beginnen. Der für Programm und Daten freie Speicherraum beträgt dann ca. 47.000 Zeichen.

1.6. Initialisierung (INIT)

Die Initialisierung— auch Formatierung genannt— bereitet die fabrikneue Leerdiskette zur Datenaufnahme vor, indem softwaremäßig ein bestimmtes Bit-Muster auf die Diskette geschrieben wird, damit der Lesekopf später die einzelnen Spuren und Sektoren finden kann (Apple-Disketten sind soft-sektoriert, d.h. softwaremäßig in Sektoren eingeteilt. Gegensatz: hard-sektoriert mit z.B. 16 Sektoren-Löchern).

Nach dem Booten einer bereits vorliegenden Programmdiskette, z.B. der „System Master"-Diskette, wird das sog. Hello-Programm geladen. Dies ist dasjenige (Applesoft-)Programm, unter dessen Name die Programmdiskette zuvor initialisiert wurde. Ein solches Hello-Programm kann z.B. so aussehen:

10 HOME : PRINT „DISKETTE ANGELEGT AM 1.3.84"

Anwender, die keinerlei Programmiererfahrung haben, verfahren bitte wie folgt:

1. „System Master"-Diskette in Laufwerk 1 einlegen
2. PR #6— gefolgt von der Return-Taste— eingeben
3. Leerdiskette in Laufwerk 1 einlegen
4. NEW— in Großbuchstaben, gefolgt von der Return-Taste— eingeben
5. 10 HOME : PRINT „DISKETTE ANGELEGT AM ..."
— gefolgt von der Return-Taste— eingeben
6. INIT ZUNAME— gefolgt von der Return-Taste— eingeben

Damit wird die Leerdiskette unter Ihrem Namen initialisiert und zur Datenaufnahme vorbereitet.

1.7. CATALOG, LOCK, UNLOCK, VERIFY

Geben Sie jetzt den Befehl CATALOG— gefolgt von Return— ein. Sie sehen dann den Catalog oder das Disketteninhaltsverzeichnis (auch Directory genannt), das etwa im Falle des Namens Meier folgendermaßen aussieht:

DISK VOLUME 254

A 002 MEIER

„A" steht für Applesoft. „002" steht für die Anzahl der Sektoren (2), die das Applesoft-Programm namens „MEIER" einnimmt. Um die Länge einer Datei überschlägig zu berechnen, multiplizieren Sie die Anzahl der Sektoren minus 1 mit 256. Hier gilt: (2 minus 1) mal 256 = 256, d.h. die Datei MEIER ist bis zu 256 Zeichen lang.

Tippen Sie nun— stets gefolgt von der Return-Taste—

LOCK MEIER

und dann

CATALOG

Sie sehen jetzt

DISK VOLUME 254

*A 002 MEIER

Vor dem „A“ steht also jetzt ein Sternchen, was besagt, daß die Datei „ge-LOCKt“ oder schreibgeschützt wurde (to lock = zuschließen). Mit dem LOCK-Befehl kann man also eine Datei vor dem Überschreiben schützen.

Tippen Sie nun

UNLOCK MEIER

und dann

CATALOG

Sie sehen dann wieder den ursprünglichen Catalog, d.h. das Sternchen wurde wieder entfernt. Mit dem UNLOCK-Befehl (to unlock = aufschließen) läßt sich also der Schreibschutz wieder aufheben.

Der LOCK-Befehl ist nur ein relativer Schutz vor dem Überschreiben einer Datei, denn wenn Sie z.B. jetzt INIT MEIER erneut eingeben würden, so würde die Diskette durch den Formatierungsvorgang völlig überschrieben, d.h. man kann auch bereits initialisierte Disketten erneut initialisieren. Wenn sie ganz sicher gehen wollen, daß die Daten auf der Diskette nicht mehr verändert werden, dann überkleben Sie die seitliche Schreibschutzeinkerbung auf der Diskette mit dem in der Disketten-Box befindlichen Klebestreifen. Dann kann die Diskette weder überschrieben noch durch Unachtsamkeit aus Versehen neu initialisiert werden.

Tippen Sie nun

VERIFY MEIER

und danach

VERIFY MUELLER

Im ersteren Fall passiert gar nichts, während Sie im letzteren Fall die Fehlermeldung FILE NOT FOUND (= Datei nicht gefunden) erhalten. Der VERIFY-Befehl (to verify = überprüfen) überprüft erstens die Existens einer Datei und zweitens wird die Intaktheit der Datei durch vollständiges Einlesen derselben ermittelt. Es gibt also die Möglichkeit, daß zwar der Dateiname aus dem Catalog ersichtlich ist, aber der VERIFY-Befehl trotzdem zu einer Fehlermeldung führt (dann meist I/O ERROR), was besagt, daß die Datei physisch defekt ist, z.B. wenn Zigarettenasche auf die Diskette gefallen ist.

1.8 Kopieren von Disketten und Dateien

Die „System Master"-Diskette enthält zwei wichtige Kopierprogramme, nämlich

1. COPYA zum Duplizieren kompletter Disketten, und
2. FID zum Kopieren einzelner Dateien.

Beide Programme sind in dem „DOS Handbuch" von Apple ausführlich beschrieben, so daß hierauf nicht näher eingegangen zu werden braucht. COPYA wird mit
RUN COPYA
gestartet, während FID mit
BRUN FID
gestartet wird.

Anstelle von COPYA kann man auch das erheblich schnellere Programm COPY.IIE verwenden, dessen Listing im Teil II dieses Buches abgedruckt ist.

2. DOS für Applesoft-Programmierer

2.1. Dateinamen

Während maschinensprachlich die meisten ASCII-Zeichen als Dateinamen möglich wären, sollte man normalerweise bei der Wahl der Dateinamen folgende Regeln beachten:

1. Ein Dateiname sollte nur aus Großbuchstaben, Ziffern, Leertaste und Punkt bestehen.
2. Ein Dateiname sollte stets mit einem Großbuchstaben beginnen.
3. Ein Dateiname darf höchstens 30 Zeichen lang sein

Danach wäre z.B. „+1 Datei" ein illegaler Dateiname, weil er mit keinem Großbuchstaben beginnt und außerdem noch Kleinbuchstaben enthält. Beispielsweise könnte der Applewriter IIe infolge eines Programmierfehlers keine Dateinamen einlesen, die den Bindestrich enthalten, z.B. „DATEI-1" (Grund: Der Applewriter versucht, den „-" in einen Großbuchstaben umzuwandeln, was natürlich nicht geht). Wenn man die obigen Regeln beachtet, entstehen niemals Konflikte bei Anwenderprogrammen.
Auf eine Diskette passen insgesamt 105 Dateinamen und damit theoretisch insgesamt 105 Dateien. Soviele Dateien können jedoch normalerweise niemals auf einer Diskette untergebracht werden, weil sonst jede Datei im Durchscnitt weniger als 5 Sektoren einnehmen müßte, denn eine Diskette enthält normalerweise nur 496 Daten-Sektoren (31 Spuren mal 16 Sektoren).

2.2. Befehlsnamen

Ähnlich wie bei Applesoft (z.B. HOME, NEW, PRINT usw.) gibt es bei DOS zahlreiche Befehlsnamen (z.B. CATALOG, LOCK, UNLOCK, VERIFY usw.). Im einzelnen sind folgende Gemeinsamkeiten und Unterschiede zu beachten:

a) Applesoft-Befehlsnamen werden speicherintern als sog. Tokens (= 1-Byte-Verschlüsselungen) gepackt gespeichert; z.B. ist der interne Code $BA (dezimal 186) das Token für PRINT. Nur während der Eingabe einer neuen Programmzeile oder während des Listens eines Programms erscheinen die Tokens „ausgeschrieben". Umgekehrt werden die DOS-Befehlsnamen als Bestandteile eines Applesoft-Programms stets „ausgeschrieben", d.h befinden sich so im Speicher, wie man sie beim Listen sieht (als ASCII-Folge mit Bit 7 off). Dafür können DOS-Befehlsnamen als String-Variablen deklariert werden, was bei Applesoft-Befehlsnamen nicht möglich ist. Beispiel:

10 L$ = „LIST" : REM nicht erlaubt!
20 C$ = „CATALOG": REM erlaubt

b) Genau wie bei Applesoft müssen auch bei DOS die Befehlsnamen in Großbuchstaben geschrieben werden, also List und Catalog falsch, LIST und CATALOG richtig. DOS-Befehlsnamen sind 2 - 8 Zeichen lang.

2.3. Direkte und indirekte DOS-Befehle

Ähnlich wie bei Applesoft gliedert man bei DOS die Befehle in zwei Gruppen:

2.3.1. Direkt- UND Indirektbefehle

Direkt- UND Indirektbefehle lassen sich SOWOHL über die Tastatur ALS AUCH aus Applesoft-Programmen (sowie Integer- und Assembler-Programmen) heraus geben.

Direktbefehl (Tastatureingabe-Modus, Nicht-RUN-Modus):

Der Tastatureingabe-Modus (immediate mode) liegt dann vor, wenn sich das „Prompt" Ü (oder beim amerikanischen Apple die eckige Klammer) am linken Bildschirmrand befindet und dahinter der „Cursor" (Schreibmarke) blinkt.

Der DOS-Direktbefehl unterscheidet sich vom Applesoft-Direktbefehl durch folgendes:

— Im Applesoft können mehrere Direktbefehle gleichzeitig in einer Zeile eingegeben werden, z.B.

HOME : PRINT 7 + 3 : LIST 10
Bei DOS-Befehlen ist dies nicht möglich; z.B. würde
CATALOG : CATALOG
zu einer Fehlermeldung führen.

— Darüber hinaus können DOS-Direktbefehle nur nach einem Return eingegeben werden, d.h. der Cursor (Ü bei Applesoft) muß sich am linken Bildschirmrand befinden.

Indirektbefehl (Programm-Modus, RUN-Modus):

Einem DOS-Indirektbefehl muß stets ein Return oder CHR$(13) und ein Ctrl-D oder CHR$(4) vorangehen und der Befehl muß in Anführungszeichen gesetzt und als PRINT-Befehl ausgeführt werden, der mit einem Return endet.

Richtige Beispiele:

10 D$ = CHR$(4) : PRINT D$;„CATALOG“

oder

10 D$ = CHR$(4) : PRINT D$„CATALOG“ : REM Semikolon zwischen D$ und „CATALOG“ darf fehlen!

oder

10 PRINT CHR$(4);„CATALOG“

oder

10 PRINT CHR$(4)„CATALOG“

oder

10 PRINT „DCATALOG“ : REM Hier steht das D für ein eingegebenes Ctrl-D

oder

10 X$ = CHR$(4) + „CATALOG“ : PRINT X$: REM Variablen-Zuweisung erlaubt!

oder

10 D$ = CHR$(4) : C$ = „CATALOG“ : PRINT D$;C$

oder

10 PRINT CHR$(4) „CATALOG“ : PRINT CHR$(4) „CATALOG“ : REM Mehrfachbefehle in einer Programmzeile erlaubt!

Falsche Beispiele:

10 PRINT „CATALOG“ : REM Ctrl-D fehlt!

oder

10 CATALOG : REM PRINT, Ctrl-D und Anführungszeichen fehlen!

oder

10 PRINT CHR$(4);„CATALOG“; : REM Semikolon nach CATALOG unterdrückt Return!

Folgende DOS-Befehle können sowohl in Direkt- wie Indirektmodus gegeben werden:

LOAD
RUN
SAVE
BLOAD
BRUN
BSAVE
CATALOG
VERIFY
RENAME
DELETE
LOCK
UNLOCK
MON
NOMON
PR # + Slot

IN # + Slot
EXEC
CLOSE

CHAIN (nur Integer-Basic!)
INIT (!)
MAXFILES (!)
FP (!)
INT (!)

Bei den mit Ausrufezeichen markierten Befehlsnamen ist Vorsicht im Programm-Modus geboten. Näheres steht bei den einzelnen Befehlserläuterungen.

2.3.2. NUR Indirektbefehle

Reine Indirektbefehle lassen sich nur aus dem Programm heraus geben. Es sind dies die Befehle:

OPEN
READ
WRITE
POSITION
APPEND

Es handelt sich hierbei ausschließlich um sog. Textfile-Befehle.

2.4. Parameter Slot, Drive, Volume

DOS-Befehle können durch sog. Parameter näher bestimmt werden. Es sind dies die allgemeinen Parameter Slot (S), Drive (D) und Volume (V) sowie einige befehlsspezifische Parameter wie L(ength) usw., die bei den einzelnen Befehlen behandelt werden. Die Reihenfolge der Parameter ist gleichgültig. Das Betriebssystem merkt sich die zuletzt definierten Parameter, so daß man die Parameter nicht ständig wiederholen muß. Beispiele:

CATALOG, S6,D1 oder
CATALOG, D1,S6

Wie ersichtlich, werden die Parameter vom Befehlsnamen durch Kommas abgegrenzt. Aus diesem Grunde darf ein Dateiname niemals ein Komma enthalten!

1. Slot = S (Steckplatz; Werte 1-7)

Der Slot-Parameter ist im Bereich 1-7 (beim Apple II Plus wegen einer möglichen RGB-Karte in Slot 7 nur im Bereich 1-6) möglich, z.B.

CATALOG, S4

2. Drive = D (Laufwerk, Werte 1-2)

Der Drive-Parameter kann 1 oder 2 sein, z.B.

CATALOG, D1
CATALOG, D2

3. Volume = V (Disketten-Nummer, Werte 0-254)

Eine Diskette kann eine Nummer erhalten, die ausschließlich beim Initialisieren festgelegt wird und nachträglich praktisch nicht mehr geändert werden kann (weil die Volume-Nummer im sog. Adreßfeld, einem Art Vorspann zu jedem 256-Byte-Sektor, beim Formatieren verankert wird.)

Der zulässige Bereich für Volume-Nummern ist 1-254, z.B.

INIT MEIER, V1
INIT MEIER, V10

Sonderfall:

INIT MEIER, V0
INIT MEIER (ohne Parameter)

ergibt stets die Volume-Nummer 254.

Der Nutzen der Volume-Nummer ist gering, da DOS nicht in der Lage ist, automatisch zu ermitteln, auf welchem Slot und Drive sich eine mit einer Volume-Nummer spezifizierte Diskette befindet. (Dies ist z.B. bei Pascal und ProDOS in bezug auf die Volume-Names möglich.)

Die Volume-Nummer spielt dagegen bei Festplattenlaufwerken (z.B. Corvus) eine große Rolle, da z.B. die Corvus in 50-100 Volumes eingeteilt ist, je nach Größe des Plattenspeichers.

Die Parameter S, D und V sind bei fast allen Befehlen möglich. Ausnahmen:

- bei CATALOG wird der V-Parameter ignoriert
- bei bestimmten Befehlen sind die Parameter weder sinnvoll noch möglich, z.B. MON, NOMON, MAXFILES
- bei CLOSE sind Parameter nicht erforderlich (und auch nicht zulässig), da dieser Befehl als einziger vollautomatisch alle Dateien auf allen Laufwerken schließen kann.
- als sehr lästig erweist es sich, daß bei den READ- und WRITE-Befehlen keine Parameter zulässig sind (wie man dies umgeht, siehe weiter unten)

Gesamtaufstellung (in Beispielen):

```
INIT,S6,D1,V1
CATALOG,S5,D2 (kein V)
RENAME ALT,NEU,S6,D1,V1
DELETE ALT,S2,D2,V100
LOCK DATEI,S6,D1,V1
UNLOCK DATEI,S6,D1,V1
VERIFY DATEI,S6,D2,V0

SAVE PROGRAMM,S6,D1,V5
LOAD PROGRAMM,S5,D2,V0
RUN HELLO,S6,D1,V254
BSAVE BILD,A$4000,L$2000,S6,D1,V1
BLOAD BILD,S6,D1,V1
BRUN MASCHINENPROGRAMM,S6,D1,V1
EXEC EXECUTIVE.FILE,S6,D2,V10
CHAIN INTEGER.PROGRAMM,S6,D1,V1

MON (kein S,D,V)
NOMON (kein S,D,V)
MAXFILES 3 (kein S,D,V)
FP (normal kein S,D,V)
INT (normal kein S,D,V)
PR#6 (nur S)
IN#2 (nur S)
```

```
10 PRINT CHR$(4);,,OPEN DATEI,S6,D1,V1"
10 PRINT CHR$(4);,,CLOSE DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,WRITE DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,READ DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,APPEND DATEI,S6,D1,V1"
10 PRINT CHR$(4);,,POSITION DATEI" (kein S,D,V)
```

2.5. Einfache DOS-Befehle

Als einfach bzw. unkompliziert können alle diejenigen DOS-Befehle bezeichnet werden, die nicht mit Textfiles zusammenhängen.

2.5.1. INIT, CATALOG, LOCK, UNLOCK, VERIFY, DELETE, RENAME

INIT dient — wie bereits geschildert — zum Initialisieren fabrikneuer oder zum Reinitialisieren bereits früher initialisierter (= formatierter) Disketten:

```
10 PRINT ,,ZU FORMATIERENDE LEERDISKETTE EINLEGEN!"
20 INPUT ,,WELCHE VOLUME-NUMMER (1-254):";X$
30 PRINT CHR$(4),,INIT DATENDISK";,, ,V";VAL(X$)
40 PRINT CHR$(4),,DELETE DATENDISK"
50 GOTO 20
```

Dieses kleine Programm würde jeweils nach der gewünschten Volume-Nummer fragen und dann die eingelegte Leerdiskette initialisieren und im Anschluß daran den Hello-Namen (= Boot-Programm-Namen) wieder löschen. Wenn sich auf einer Diskette kein Hello-Programm mehr befindet, bootet die Diskette trotzdem korrekt, doch erfolgt dann die Meldung „FILE NOT FOUND", weil dieses Programm ja jetzt nicht mehr existiert.

Wie aus diesem Beispiel ersichtlich, ist es zwar möglich, aus einem Programm heraus zu initialisieren, doch sollte man unbedingt eine Warnmeldung einbauen, damit nicht aus Versehen eine wertvolle Datendiskette durch die Initialisierung zerstört wird.

Die Befehle CATALOG, LOCK, UNLOCK und VERIFY sind bereits bekannt.

Auch die Befehle RENAME (Dateiname umbenennen) und DELETE (Datei löschen) verstehen sich von selbst. Die Syntax ist bei RENAME:

RENAME ALTER NAME, NEUER NAME

RENAME ist im DOS nicht narrensicher implementiert mit der Folge, daß es möglich ist, mit RENAME zwei gleiche Dateinamen auf der Diskette zu erzeugen. Nehmen wir an, die Diskette enthalte bereits die Files

PROGRAMM.1
PROGRAMM.2

Mit RENAME PROGRAMM.1,PROGRAMM.2 würde man dann auf der Diskette zwei gleiche Dateinamen erzeugen:

PROGRAMM.2
PROGRAMM.2

Da die LOAD-, BLOAD-Befehle usw. die Suche nach dem zuerst gefundenen Dateinamen abbrechen, würde man das ehemalige, eigentliche PROGRAMM.2 niemals mehr einladen können. Man muß dann schleunigst den oberen Dateinamen mit RENAME PROGRAMM.2,PROGRAMM.1 wieder zurückbenennen.

Demo-Programm:

```
100 HOME : INPUT „DISKETTE EINLEGEN “;X$
110 PRINT CHR$(4),„SAVE XXX“
120 PRINT CHR$(4),„CATALOG“
130 PRINT CHR$(4),„LOCK XXX“
140 PRINT CHR$(4),„CATALOG“
150 PRINT CHR$(4),„UNLOCK XXX“
160 PRINT CHR$(4),„CATALOG XXX“
170 PRINT CHR$(4),„RENAME XXX,YYY“
180 PRINT CHR$(4),„CATALOG“
190 PRINT CHR$(4),„DELETE YYY“
200 PRINT CHR$(4),„CATALOG“
210 PRINT „JETZT KOMMT 'FILE NOT FOUND'-MELDUNG“
220 PRINT CHR$(4),„VERIFY YYY“
```

Warum kommt am Ende dieses Programmes eine Fehlermeldung?

2.5.2. FP und INT

FP steht für Floating Point Basic (= Fließkomma-Basic = Applesoft) und INT steht für Integer Basic (= Ganzzahl-Basic). Der Apple IIe sowie der Apple II Plus haben Applesoft im ROM (im Bereich $D000-$FFFF), während der Uralt-Apple II Integer-Basic im ROM hatte. Die „System Master"-Diskette lädt automatisch nach dem Booten Integer-Basic in die Language Card (RAM-Bereich $D000-$FFFF). Mit INT kann man dann zu Integer umschalten, und mit FP wieder zu Applesoft zurückschalten. Da Integer-Basic meistens nicht mehr benutzt wird, hat der FP-Befehl heute eine andere Bedeutung. Wenn man sich im Applesoft-Modus befindet und FP eingibt, dann wird

1. die Speicherstelle 2048 (hexadezimal $0800) gelöscht (Dies macht NEW nicht. Wäre hier ein anderer Wert außer 0, dann würde RUN nicht mehr funktionieren.)
2. HIMEM (Highest Memory = Obergrenze des freien Speichers) auf 38400 (hexadezimal $9600) zurückgesetzt, womit gleichzeitig MAXFILES 3 eingestellt wird, und
3. der NEW-Befehl ausgeführt.

FP ist damit wirkungsvoller als NEW allein. FP ist praktisch der einzige Befehl, der HIMEM normalisiert. (Falls sich das DOS selbst in der Language Card befindet, bewirkt FP in der Regel HIMEM $BF00.) Der String-Pool von Applesoft beginnt übrigens bei HIMEM-1, also bei $95FF.

FP und INT als Indirektbefehle bewirken den sofortigen Programmabbruch und das Löschen des jeweiligen Programms, z.B.

```
1000 PRINT CHR$(4),„FP"
```

2.5.3. MAXFILES

MAXFILES heißt „maximum number of files" = maximale Puffer- und damit Dateianzahl. Für jede geöffnete Datei wird ein 595 Bytes umfassender Zwischenpuffer angelegt. Die normalerweise 3 automatisch nach dem Booten angelegten Puffer befinden sich in dem Bereich

$9600-$9CF8 (dezimal 38400-40184); (40184-38400 + 1) : 595 = 3

Wenn Befehle wie CATALOG, LOAD, VERIFY usw. ausgeführt werden, wird vorübergehend 1 Puffer benötigt. Der zuerst belegte Puffer ist der Puffer ab $9600.

Nach Beendigung des CATALOG-Befehls usw. wird der Puffer wieder frei. Mit dem Befehl MAXFILES + Zahl, z.B.

MAXFILES 5

kann man die Anzahl der Puffer festlegen. Es sind maximal 16 Puffer möglich, wobei z.B. bei MAXFILES 4 HIMEM um 595 Bytes niedriger gesetzt wird als $9600, d.h. je mehr Puffer, desto weniger freier Speicherraum. (Wenn DOS in die Language Card verschoben wurde, hat man in der Regel nur maximal 5 Puffer zur Verfügung, die automatisch ständig benutzt werden können, so daß hier der MAXFILES-Befehl wirkungslos ist.)
Im einzelnen gilt für normales DOS in den unteren 48K:

MAXFILES 1 = HIMEM: 39590 = $9AA6
MAXFILES 2 = HIMEM: 38995 = $9853
MAXFILES 3 = HIMEM: 38400 = $9600 (Normalfall)

Der momentane Wert von HIMEM läßt sich übrigens ermitteln mit:

PRINT PEEK (115) + PEEK (116) · 256

MAXFILES darf aus einem Programm heraus nur dann geändert werden, wenn zuvor entweder keine Strings definiert wurden oder der Variablen-Speicher mit CLEAR gelöscht wurde. MAXFILES definiere man deshalb am besten als erste Zeile des Applesoft-Programms, z.B.:

```
10 CLEAR : PRINT CHR$(4),,MAXFILES 1"
```

MAXFILES spielt insbesondere bei Textfiles eine Rolle, da für jeden durch OPEN geöffneten Textfile ein Puffer eingerichtet wird, z.B.:

```
10 CLEAR : PRINT CHR$(4),,MAXFILES 4"
20 PRINT CHR$(4),,OPEN DATEI.1"
30 PRINT CHR$(4),,OPEN DATEI.2"
40 PRINT CHR$(4),,OPEN DATEI.3"
50 PRINT CHR$(4),,OPEN DATEI.4"
60 PRINT ,,Achtung:"
70 PRINT ,,Jetzt kommt Meldung"
80 PRINT ,,NO BUFFERS AVAILABLE"
90 PRINT CHR$(4),,CATALOG"
```

Da hier nur 4 Puffer angelegt wurden, war kein Puffer mehr für den CATALOG-Befehl zur Verfügung.

2.5.4. MON und NOMON

MON kommt von „to monitor" = überwachen. MON bedeutet DOS-Befehle überwachen, NOMON DOS-Befehle nicht mehr überwachen. Mit Überwachen ist gemeint, daß bei einem laufenden Programm die DOS-Befehle sowie bei Textfiles darüber hinaus die eingelesenen oder gespeicherten Daten am Bildschirm sichtbar werden. Es gibt folgende spezifische Parameter:

C = Command = DOS-Befehle werden sichtbar
I = Input = eingelesene Textfiles werden sichtbar (während des Einlesens)
O = Output = gespeicherte Textfiles werden sichtbar (während des Speicherns)

Beispiele:

MON C,I,O
NOMON C,I,O (auch NOMONCIO, d.h. Kommas entbehrlich)
MON C
NOMON C
usw.

Man beachte, daß die Befehle MON und NOMON ohne Parameter wirkungslos sind.

2.5.5. PR # S und IN # S

Diese Befehle steuern den Output (PR # + Slot) und Input (IN # + Slot) von Peripheriegeräten. Beispiele:

- mit PR# 1 wird der Drucker aktiviert, falls in Slot 1 ein Drucker-Interface steckt und der Drucker bereits eingeschaltet wurde, d.h. auf ON LINE steht.
- mit PR#6 oder auch IN#6 wird das DOS gebootet, falls sich ein DOS-Controller in Slot 6 befindet.
- mit PR # 3 wird die 80-Zeichen-Karte eingeschaltet, falls sich eine solche in Slot 3 befindet, usw.

Aus dem Programm heraus sollten diese Befehle steht mit Ctrl-D ausgeführt werden,

da sonst die sog. DOS-Vektoren und damit das DOS „abgehängt“ werden.

```
10 PRINT CHR$(4),„PR#1“
```

würde den Drucker einschalten und gleichzeitig das DOS angeschlossen lassen.

```
10 PR#0 : IN#0 : PR#1
```

würde den Drucker zwar ebenfalls einschalten, aber das DOS komplett abhängen.

```
10 PR#0 : IN#0 : CALL 1002
```

würde dann nach Beendigung des Druckvorgangs das DOS wieder ordnungsgemäß „anhängen“.
Das planmäßige, vorübergehende „Abhängen“ des DOS mit PR#0 : IN#0 und das spätere „Anhängen“ mit PR#0 : IN#0 : CALL 1002 hat den Vorteil, daß die Bildschirm- und Druckergeschwindigkeit während des abgeschalteten DOS größer ist. Weiteres Beispiel:

```
10 PR# 0 : IN#0 : REM DOS ABGEHÄNGT
20 FOR X = 1 TO 10
30 PRINT „AAAAAAAAAA“
40 NEXT X
50 PR#1
60 FOR X = 1 TO 10
70 PRINT „AAAAAAAAAA“: NEXT X
80 PR#0 : IN#0
90 PRINT CHR$(4) „CATALOG“ : REM WIRKUNGSLOS
100 PR#0 : IN#0 : CALL 1002 : PRINT CHR$(4) „CATALOG“
```

2.5.6. LOAD, RUN, SAVE (Basicfiles)

LOAD, RUN und SAVE sind die DOS-Befehle zum Laden (= LOAD), Laden UND Starten (= RUN) sowie zum Speichern (= SAVE) von Applesoft- und Integer-Programmen (= Basicfiles). Die Handhabung dieser Befehle ist denkbar einfach, da das DOS die Berechnung der Länge des jeweiligen Programms selbst übernimmt. Beispiele:

```
LOAD HELLO
```

```
RUN HELLO
SAVE HELLO

10 PRINT „DIES IST PROGRAMM.1“
1000 PRINT CHR$(4) „RUN PROGRAMM.2“
```

Programme lassen sich aus einem Program heraus starten. Allerdings gehen dadurch alle Variablen von Programm.1 verloren.

```
10 PRINT „DIES IST EIN PROGRAMM, DAS SICH SELBST SPEICHERT“
20 PRINT CHR$(4) „SAVE PROGRAMM“
```

Programme lassen sich aus dem Programm heraus selbst speichern, doch ist dies normalerweise nicht besonders sinnvoll.

Basic-Programme werden normalerweise ab 2048 + 1 (= $0800 + 1) in den RAM-Speicher geladen. Start- und Endadresse eines Applesoft-Programms kann man bei Bedarf manuell ermitteln durch folgende Formeln:

Startadresse: PRINT PEEK (103) + PEEK (104) · 256
Endadresse: PRINT PEEK (175) + PEEK (176) · 256

Auf der Diskette werden Basic-Programme als Speicherauszüge ab $0801 bis zum Programmende inclusive 4 Hex-Codes $00 nach dem Programmende und mit einem 2 Bytes langen Vorspann, der die Länge beinhaltet (in hexadezimal, Low Byte — High Byte), gespeichert. (Nach den 4 Hex $00 wird infolge eines DOS-Fehlers 1 weiteres Byte gespeichert.)

Basicfiles sind aus dem Catalog wie folgt ersichtlich:

A steht für Applesoft
I steht für Integer

Dagegen steht B für Binärfile und T für Textfile.

2.5.7. BLOAD, BRUN und BSAVE (Binärfiles)

Diese Befehle beziehen sich auf sog. Binärfiles, d.h. auf Maschinenprogramme sowie reine Speicherauszüge, z.B. Hires-Bilder, Grafik-Shapes usw. BLOAD steht für bi-

näres Laden, BRUN für binäres RUN (ausschließlich von Maschinenprogrammen) und BSAVE steht für binäres Speichern.
Der BSAVE-Befehl verlangt als Parameter die Startadresse (A) sowie die Länge (L) in entweder dezimaler oder hexadezimaler Schreibweise (auch gemischt möglich), z.B.:

BSAVE BILD, A$2000, L$2000 oder
BSAVE BILD, A8192, L8192 oder
BSAVE BILD, A$2000, L8192 oder
BSAVE BILD, A8192, L$2000

Angenommen, im RAM ab $0300 sei Nachstehendes gespeichert:

0300: 01 02 03 04 05 06

Wollte man dies auf der Diskette speichern, so würde folgender Befehl genügen:

BSAVE TEST,A$0300,L$0006 oder kürzer
BSAVE TEST,A$300,L$6 (Führende Nullen können weggelassen werden.)

Der entsprechende Disketten-Sektor würde dann so aussehen:

00 03 06 00 01 02 03 04 05 06

Die ersten zwei Bytes sind die Startadresse (00 03 — 0300 — $0300 — also zuerst Low Byte, dann High Byte)
Die nächsten zwei Bytes sind die Länge (06 00 — 00 06 — $0006 — wieder Low Byte first)

Für vierstellige, hexadezimale Zahlen im Bereich $0000-$FFFF gilt folgende Umrechnungsformel:

Low Byte + (High Byte · 256)

Mit BLOAD wird der Binärfile genau an diejenige Stelle eingeladen, die beim vorangehenden BSAVE spezifiziert wurde. Man kann einen Binärfile allerdings auch an eine andere Stelle bloaden durch Spezifikation einer neuen Startadresse. Leider gibt es beim DOS keinen Befehl, der die Startadresse und Länge eines Binärfiles anzeigt, so daß man mit den folgenden Formeln die Berechnung selbst vornehmen muß:

BLOAD-Startadresse: PRINT PEEK (43634) + PEEK (43635) · 256
BLOAD-Länge: PRINT PEEK (43616) + PEEK (43617) · 256 (vgl. S. 67!)

Wenn das DOS in die Language Card verschoben wurde, wird in der Regel anstelle des INIT-Befehls ein entsprechender Befehl implementiert, z.B. PAD bei Diversi-DOS.

Der BRUN-Befehl darf nur bei Maschinenprogrammen angewandt werden. BRUN lädt zunächst den Binärfile und springt dann zur Startadresse.

Die Länge eines Binärfiles darf 32767 (= $7FFF) normalerweise nicht überschreiten. Die Startadresse kann im Bereich 0-65535 ($0000-$FFFF) liegen.

Als freie Bereiche für Binärfiles kommen in Frage:

$0200-$02FF (nur für vorübergehend benötigte Startprogramme, da dies der Tastatureingabepuffer ist)
$0300-$03CF
$0803-$095FF (Maschinenprogramme sollte man besser ab $803 statt $800 beginnen lassen, da Applesoft dann „denkt", daß kein Programm geladen wurde.)

Wenn sich DOS in der Language Card befindet, kann man auch noch den Bereich $9600-$BEFF benutzen. Umgekehrt, wenn sich DOS nicht in der Language Card befindet, stehen die oberen 16K des RAM-Speichers ebenfalls für Maschinenprogramme zur Verfügung.

Zusammenfassend gilt für Applesoft-, Integer- und Binär-Files folgender „Vorspann" auf dem ersten Datensektor des Files auf der Diskette:

1. Integer: Low Byte + High Byte der Länge,
danach Programm,
danach 4 Hex-Codes 00

2. Applesoft: Low Byte + High Byte der Länge,
danach Programm,
danach 4 Hex-Codes 00

3. Binärfile: Low Byte + High Byte der Startadresse,
Low Byte + High Byte der Länge,
danach eigentlicher Binärfile

2.5.8. CHAIN

Der CHAIN-Befehl sei hier nur der Vollständigkeit halber erwähnt. Er gilt nur für das kaum noch benutzte Integer-Basic und dient dem Zweck, aus einem Integer-Programm heraus ein weiteres Integer-Programm mit RUN zu starten, ohne daß dadurch die Variablen zerstört werden, z.B.:

1000 PRINT „D CHAIN ZWEITPROGRAMM" : REM D = CTRL-D

Für Applesoft befindet sich auf der „System Master"-Diskette eine ähnliche Maschinen-Routine, die jedoch nicht immer funktioniert. Auch bei Applesoft-Compilern wie TASC usw. funktioniert das „Chaining" nur teilweise. Für Besitzer von RAM-Karten, z.B. der Apple IIe 64K Karte, gibt es eine viel sicherere Lösung. Man speichert die Variablen auf der schnellen RAM-Disk, startet das nächste Programm-Modul und lädt schließlich die zwischengespeicherten Variablen von der RAM-Karte wieder zurück. Ein RAM-Disk-Driver ist im Teil II dieses Buches als Listing abgedruckt.

2.6. Textfile-Befehle

2.6.1. OPEN, READ, WRITE, CLOSE

Informationstheoretisch gesehen gibt es zwei grundsätzlich verschiedene Arten von Dateien, nämlich

1. Programm-Dateien, z.B. Applesoft-, Integer- und Assemblerprogramme, sowie
2. Daten-Dateien, z.B. Adressen, Briefe, Bilder, Meßwerte usw.

Basicfiles (A, I) sind stets Programm-Dateien. Binärfiles sind demgegenüber teils Programm-Dateien (z.B. Assemblerprogramme) und teils Daten-Dateien, z.B. Hires-Bilder, binär abgespeicherte Zahlen-Arrays usw.

Die typische und zugleich wichtigste Daten-Datei für sowohl Applesoft- wie auch Maschinen-Programme ist der sog. Textfile. Textfile bedeutet wörtlich Text-Datei (String-Datei, Zeichenketten-Datei), doch können Textfiles auch Zahlen-Dateien beinhalten.

Beispiel: Strings

AAAAA
BBBBB

```
10 PRINT CHR$(4) „OPEN STRINGS“
20 PRINT CHR$(4) „WRITE STRINGS“
30 PRINT „AAAAA“
40 PRINT „BBBBB“
50 PRINT CHR$(4) „CLOSE“
```

```
10 PRINT CHR$(4) „OPEN STRINGS“
20 PRINT CHR$(4) „READ STRINGS“
30 INPUT A$
40 INPUT B$
50 PRINT CHR$(4) „CLOSE“
```

Beispiel: Zahlen

11111
22222

```
10 PRINT CHR$(4) „OPEN ZAHLEN“
20 PRINT CHR$(4) „WRITE ZAHLEN“
30 PRINT 11111
40 PRINT 22222
50 PRINT CHR$(4) „CLOSE“
```

```
10 PRINT CHR$(4) „OPEN ZAHLEN“
20 PRINT CHR$(4) „READ ZAHLEN“
30 INPUT Z1
40 INPUT Z2
50 PRINT CHR$(4) „CLOSE“
```

Aus den Beispielen können wir folgendes ersehen:

1. Ein Textfile, gleichviel ob er neu angelegt wird oder bereits existiert, d.h. bereits früher angelegt wurde, wird mit OPEN + DATEINAME geöffnet. Wenn eine Datei bereits angelegt war und nunmehr durch neue Werte überschrieben werden soll, so empfiehlt es sich, falls der neue Datei-Inhalt kleiner als der alte

wird, die Altdatei zunächst zu löschen, da sonst die Neudatei auf der Diskette den gleichen Raum einnehmen wird wie die Altdatei. Beispiel:

```
10 REM ALTDATEI MIT 1000 STRINGS
20 PRINT CHR$(4) „OPEN TEST“
30 PRINT CHR$(4) „WRITE TEST“
40 FOR X = 1 TO 1000 : PRINT „AAAAA“ : NEXT X
50 PRINT CHR$(4) „CLOSE“
```

```
10 REM NEUDATEI MIT 500 STRINGS
20 PRINT CHR$(4) „OPEN TEST“
30 PRINT CHR$(4) „DELETE TEST“
40 PRINT CHR$(4) „OPEN TEST“
50 PRINT CHR$(4) „WRITE TEST“
60 FOR X = 1 TO 500 : PRINT „AAAAA“ : NEXT X
70 PRINT CHR$(4) „CLOSE“
```

Bemerkungen:

a) Die Zeile 20 bei dem Neudatei-Programm kann entfallen, falls man sicher ist, daß sich auf der Diskette bereits eine Datei dieses Namens befindet. Ist man sich jedoch unsicher, ob die TEST-Datei bereits existiert, dann kann man mit OPEN – DELETE – OPEN die „FILE NOT FOUND“-Fehlermeldung umgehen.

b) Hätte man die Neudatei nicht zunächst gelöscht, dann würde die Neudatei weiterhin auf der Diskette denselben Raum wie die Altdatei einnehmen, weil DOS nach dem CLOSE die nunmehr überflüssigen, restlichen Altdatei-Datensektoren nicht von selbst löscht.

2. Nach dem OPEN-Befehl kann wahlweise ein WRITE- oder READ-Befehl folgen, je nachdem, ob auf die Datei geschrieben oder von ihr gelesen werden soll.

3. Eine Datei wird normalerweise mit PRINT-Befehlen beschrieben und mit INPUT-Befehlen gelesen.

4. Eine Datei muß mit dem CLOSE-Befehl stets geschlossen werden, auch dann, wenn nur von ihr gelesen wurde. Zwar gehen bei Dateien, aus denen nur gelesen wird, keine Daten auf der Diskette verloren, wenn der CLOSE-Befehl vergessen wurde, doch bleibt dann der für diese Datei bereitgestellte DOS-Puffer belegt. Öffnet man z.B. bei Maxfiles 3 den vierten Textfile, ohne zuvor eine der

ersten drei geöffneten Textfiles wieder zu schließen, folgt die Meldung „NO BUFFERS AVAILABLE" (= keine Puffer mehr frei). Ein fehlender CLOSE-Befehl kann auch nach Beendigung des Programms manuell nachgeholt werden, indem man über die Tastatur CLOSE tippt. CLOSE allein schließt ALLE noch offenen Dateien, CLOSE + DATEINAME schließt nur diese letztere Datei. Beispiel:

```
10 PRINT CHR$(4) „OPEN DATEI.1"
15 PRINT 1
20 PRINT CHR$(4) „OPEN DATEI.2"
25 PRINT 2
30 PRINT CHR$(4) „OPEN DATEI.3"
35 PRINT 3
40 PRINT CHR$(4) „CLOSE DATEI.1" : REM Schließt nur diese Datei.1
```

Was aus den Beispielen nicht unmittelbar evident, jedoch für die Korrektheit der Textfile-Befehle erforderlich ist, ist der Umstand, daß die OPEN-, READ- und WRITE-Befehle nur dann wirksam werden, wenn ihnen nicht nur ein Ctrl-D, sondern auch ein Ctrl-M bzw. Return vorausgeht.

Falsches Beispiel 1:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 PRINT „AAAAA"; : REM Semikolon!
40 PRINT CHR$(4) „CLOSE"
```

Kommentar: Der letzte PRINT-Befehl vor dem CLOSE-Befehl darf nicht mit einem Semikolon abschließen, da sonst der CLOSE-Befehl nicht ausgeführt wird. Beim Beispiel 1 würde auf die Diskette statt „AAAAA" in Wirklichkeit „AAAAA-CLOSE" geschrieben, d.h. Ctrl-D + „CLOSE" würde an den String „AAAAA" angehängt werden!

Richtiges Beispiel 1:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 FOR X = 1 TO 3 : PRINT „AAAAA"; : NEXT X
40 PRINT : PRINT CHR$(4) „CLOSE"
```

Kommentar: Mehrere PRINT-Befehle mit Semikolon zur Unterdrückung des Returns sind zulässig, doch muß der letzte PRINT-Befehl ein Return beinhalten.

Falsches Beispiel 2:

```
10 PRINT ,,SPEICHERN J/N ";: GET X$
20 IF X$ = ,,N" THEN END
30 PRINT CHR$(4) ,,OPEN TEST"
usw.
```

Kommentar: Der GET-Befehl unterdrückt das Return mit der Folge, daß ,,OPEN TEST" am Bildschirm angezeigt, aber nicht vom DOS als Befehl erkannt wird.

Richtiges Beispiel 2:

```
10 PRINT ,,SPEICHERN J/N "; : GET X$
20 IF X$ = ,,N" THEN END
30 PRINT : PRINT CHR$(4) ,,OPEN TEST"
```

Kommentar: Hier steht in Zeile 30 ein zusätzliches PRINT, so daß der DOS-Syntax Genüge getan wird.

2.6.2. Struktur eines Textfiles

Wenn auf die Diskette folgende Datei geschrieben wird

```
10 PRINT CHR$(4) ,,OPEN FILE"
20 PRINT CHR$(4) ,,WRITE FILE"
30 PRINT ,,aaaaa"
40 PRINT ,,bbbbb"
50 PRINT ,,ccccc"
60 PRINT CHR$(4) ,,CLOSE"
```

dann hat diese Datei die Struktur:

aaaaa	+ Return	Feld 1
bbbbb	+ Return	Feld 2
ccccc	+ Return	Feld 3
	+ Ctrl-0	Endmarker

Durch Return abgegrenzte ASCII-Folgen (Strings oder Zahlen) werden als Felder bezeichnet.

Return hat den ASCII-Code 13 oder 141 (hexadezimal $0D oder $8D) und Ctrl-0 hat den ASCII-Code 0 oder 128 (oder hexadezimal $00 oder $80). ASCII ist die Abkürzung für American Standard Code for Information Interchange, d.h. amerikanischer Datenaustausch-Code. Die Tabelle im Anhang zu diesem Buch enthält eine erweiterte ASCII-Code-Aufstellung mit den entsprechenden hexadezimalen, binären und dezimalen Code-Werten. Bereits ein flüchtiger Blick auf diese Tabelle genügt um festzustellen, daß die Tabelle sozusagen doppelt belegt ist, d.h. die Werte von 0-127 werden durch die Werte von 128-255 wiederholt. Bei der ersten Hälfte ist das linke Bit der 8-Bit-Binärverschlüsselung stets 0, bei der zweiten Hälfte stets 1. Dieses linke Bit wird Bit 7 genannt, da die Bits von rechts nach links — beginnend mit 0 — gezählt werden. Beispiel:

76543210 = Bit-Positionen
11110000 = $F0 = p

Insgesamt gibt es also 2 mal 128 verschiedene Zeichen bzw. Codes. Während Applesoft im RAM-Speicher Strings in der Regel mit „Bit 7 off" (Bit 7 = 0; linke Hälfte) speichert, werden Strings auf der Diskette in der Regel mit „Bit 7 on" (Bit 7 = 1; rechte Hälfte) gespeichert. Return hat damit den Wert 141 (oder $8D). Dagegen wird als Ausnahme Ctrl-0 auf der Diskette stets als 0 (oder $00 oder 00000000) gespeichert. Ctrl-0 (auch NUL genannt) ist der Endmarker (= die Endmarke = die Endmarkierung) für einen Textfile. Symbolisch würde die obengenannte Datei auf der Diskette also wie folgt gespeichert:

a a a a a R b b b b b R c c c c c R 0

R = Return = Ctrl-M
0 = Ctrl-0 = NUL

Und hexadezimal verschlüsselt würde die Datei so aussehen:

E1 E1 E1 E1 E1 8D E2 E2 E2 E2 E2 8D E3 E3 E3 E3 E3 8D 00

(Die Leertasten sind in beiden Fällen nur aus Gründen der besseren Lesbarkeit eingefügt.)

Bei der ASCII-Tabelle sind die Werte 160-254 (oder $A0-$FE) die sogenannten sichtbaren oder druckbaren Zeichen:

$A0-$AF:	diverse Sonderzeichen
$B0-$B9:	Ziffern
$BA-$C0:	diverse Sonderzeichen
$C0-$DA:	großes Alphabet
$DB-$E0:	länderspezifische Zeichen (Umlaute)
$E1-$FA:	kleines Alphabet
$FB-$FE:	länderspezifische Zeichen (Umlaute)

Daneben gibt es noch die Ctrl-Zeichen = Kontroll-Zeichen = Steuerzeichen = normalerweise nicht sichtbare und keinesfalls druckbare Zeichen:

$80-$9F:	diverse Kontroll-Zeichen
$FF:	DEL-Zeichen (Rückwärts-Löschzeichen)

ferner

$00:	Endmarker für Textfiles

Von den Kontroll- oder Steuerzeichen haben zwei für DOS eine besondere Bedeutung:

1. Ctrl-D = $84 = DOS-Befehlserkennungszeichen
2. Ctrl-M = $8D= Feldbegrenzungszeichen

Ctrl-D und Ctrl-M sollten möglichst niemals Bestandteil eines Strings sein, da dies sonst zu Schwierigkeiten bei Applesoft-Programmen führt.

Zahlen = Strings auf der Diskette?

Textfiles können sowohl Zahlen wie Strings enthalten. Zahlen werden jedoch auf der Diskette genauso wie Strings gespeichert, d.h. genau in der Form, wie man Zahlen am Bildschirm sieht und nicht etwa in binär verschlüsselter Form, wie sie Applesoft speicherintern abgelegt (in sog. gepackter 5-Byte-Verschlüsselung bei Fließkommazahlen und gepackter 2-Byte-Verschlüsselung bei Ganzzahlen). Beispiel:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 PRINT „12345" : REM DIES IST EIN STRING!
40 PRINT CHR$(4) „CLOSE"
50 PRINT CHR$(4) „OPEN TEST"
```

```
60 PRINT CHR$(4) „READ TEST“
70 INPUT Z
80 PRINT CHR$(4) „CLOSE“
```

Hier wird „12345“ als String gespeichert und als Zahl durch INPUT Z eingelesen.

```
10 PRINT CHR$(4) „OPEN TEST“
20 PRINT CHR$(4) „WRITE TEST“
30 PRINT 12345 : REM DIES IST EINE ZAHL!
40 PRINT CHR$(4) „CLOSE“
50 PRINT CHR$(4) „OPEN TEST“
60 PRINT CHR$(4) „READ TEST“
70 INPUT S$
80 PRINT CHR$(4) „CLOSE“
```

Hier wird 12345 als Zahl gespeichert und als String durch INPUT S$ eingelesen.

Diese beiden Beispiele machen deutlich, daß man als Strings gespeicherte Zahlen als Zahlen-Variablen einlesen kann und umgekehrt. Voraussetzung ist natürlich, daß die Strings nur Zahlzeichen enthalten, also die Ziffern von 0-9 sowie Dezimalpunkt, Plus- oder Minuszeichen sowie E (= Exponent-Zeichen).

2.6.3. Komma und Semikolon: PRINT und INPUT

An die WRITE- und READ-Befehle schließen sich üblicherweise die entsprechenden PRINT- und INPUT-Befehle an. Um mit Textfiles effektiv umgehen zu können, muß man sich zunächst einmal mit den PRINT- und INPUT-Befehlen, die eigentlich Applesoft-Befehle sind, vertraut machen.
PRINT + Zahl oder PRINT + String, z.B.
PRINT A : PRINT A$ oder
PRINT 1 : PRINT „AAAAA“
bewirken normalerweise die Bildschirmanzeige der Zahlen bzw. Strings. Ist ein Drucker angeschlossen (mit PR#1), werden die PRINT-Werte gleichzeitig an den Drucker geschickt. Ist jedoch eine DOS-Datei geöffnet, dann bewirken die PRINT-Befehle das Speichern der Werte auf der Diskette.

Nach der Basic-Syntax lassen sich 3 PRINT-Arten unterscheiden:

1. PRINT + Return, z.B. PRINT A

2. PRINT + Semikolon, z.B. PRINT A;
3. PRINT + Komma, z.B. PRINT A,

Das Semikolon am Ende eines PRINT-Statements unterdrückt das Return und das Komma am Ende eines PRINT-Statements bewirkt eine Tabulierung am Bildschirm.

Der INPUT-Befehl ist normalerweise für die manuelle Tastatureingabe aus einem Applesoft-Programm heraus gedacht. Ist jedoch eine DOS-Datei geöffnet, dann bewirkt der INPUT-Statement das Einlesen eines Strings oder einer Zahl von der Diskette und die Zuweisung zu der spezifizierten Variablen. Während PRINT statt Variablen auch Konstanten als Parameter haben kann, z.B.

PRINT 1 statt PRINT A, wobei A = 1 oder
PRINT „AAA" statt PRINT A$, wobei A$ = „AAA"

muß INPUT als Parameter stets eine Variable haben, also

INPUT A oder INPUT A$

Nach der Basic-Syntax lassen sich 2 INPUT-Arten unterscheiden:

1. INPUT + Return, z.B. INPUT A : INPUT B : INPUT C
2. INPUT + Komma, z.B. INPUT A, B, C

Für den ersten Fall gibt es bei Applesoft-Programmen entweder die Möglichkeit, daß mehrere INPUT-Statements — getrennt durch Doppelpunkt — in derselben Programmzeile stehen

```
10 INPUT A : INPUT B : INPUT C
```

oder daß die INPUT-Statements auf mehrere Programmzeilen verteilt werden

```
10 INPUT A
20 INPUT B
30 INPUT C
```

Im zweiten Fall müssen die Variablen alle in einer einzigen Zeile stehen und durch Return bei der letzten Variablen abgeschlossen werden:

```
10 INPUT A, B, C
```

2.6.4. Komma bei Mehrfachfeldern

Da PRINT-Befehle mit Kommas eigentlich eine Tabulierung bewirken, die natürlich auf der Diskette nichts zu suchen hat, sollte man die READ- und WRITE-Befehle nach meinen Erfahrungen am besten gar nicht mit Kommas implementieren, da man sonst wunderliche Überraschungen erleben kann.

Falsches Beispiel 1:

```
10 PRINT CHR$(4) „OPEN TEST“
20 PRINT CHR$(4) „WRITE TEST“
30 PRINT 1, 2, 3
40 PRINT CHR$(4) „CLOSE“
50 PRINT CHR$(4) „OPEN TEST“
60 PRINT CHR$(4) „READ TEST“
70 INPUT A, B, C
80 PRINT CHR$(4) „CLOSE“
```

Kommentar: Auf der Diskette entsteht eine Datei mit der Struktur

123R

d.h. die Ziffern sind nicht durch Kommas voneinander abgetrennt, so daß insgesamt nur eine einzige Zahl 123 auf der Diskette gespeichert wird mit der Folge, daß bei Einlesen nach der Variablen A für B und C keine Werte mehr zur Verfügung stehen.

Falsches Beispiel 2:

```
10 PRINT CHR$(4) „OPEN TEST“
20 PRINT CHR$(4) „WRITE TEST“
30 PRINT 1 „,“ 2 „,“ 3
40 PRINT CHR$(4) „CLOSE“
50 PRINT CHR$(4) „OPEN TEST“
60 PRINT CHR$(4) „READ TEST“
70 INPUT A : INPUT B : INPUT C
80 PRINT CHR$(4) „CLOSE“
```

Kommentar: Auf der Diskette entsteht eine Datei mit der Struktur

1,2,3R

d.h. die Ziffern sind durch Kommas getrennt. Da jedoch der nackte INPUT-Statement (ohne folgendes Komma) ein Return und kein Komma erwartet, bewirkt Beispiel 2 ebenso wie Beispiel 1 eine DOS-Fehlermeldung.

Richtiges Beispiel 3a:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 PRINT 1 „,“ 2 „,“ 3
40 PRINT CHR$(4) „CLOSE"
50 PRINT CHR$(4) „OPEN TEST"
60 PRINT CHR$(4) „READ TEST"
70 INPUT A, B, C
80 PRINT CHR$(4) „CLOSE"
```

Richtiges Beispiel 3b:

```
10 A = 1 : B = 2 : C = 3
20 PRINT CHR$(4) „OPEN TEST"
30 PRINT CHR$(4) „WRITE TEST"
40 PRINT A „,“ B „,“ C
50 PRINT CHR$(4) „CLOSE"
60 PRINT CHR$(4) „OPEN TEST"
70 PRINT CHR$(4) „READ TEST"
80 INPUT A, B, C
90 PRINT CHR$(4) „CLOSE"
```

Kommentar: Bei den Beispielen 3a und 3b entstehen wie bei dem Beispiel 2 auf der Diskette Files mit der Struktur

1,2,3R

Die in Anführungszeichen gesetzen Kommas bewirken, daß diese Kommas auch auf der Diskette mitgespeichert werden (im Gegensatz zum Beispiel 1, wo die Kommas nicht in Anführungszeichen gesetzt wurden). Liest man nun diese Datei mit Komma-INPUT-Statements ein, erfolgt eine korrekte Wertzuweisung zu den Variablen A, B und C. Es ist also möglich, mit Kommas zu arbeiten, doch ist das folgende Programm erheblich narrensichere, zumal dann eine einheitliche File-Struktur „Feld + Return“ stets gewährleistet wird:

```
10 PRINT CHR$(4) „OPEN TEST“
20 PRINT CHR$(4) „WRITE TEST“
30 PRINT 1 : PRINT 2 : PRINT 3
40 PRINT CHR$(4) „CLOSE“
50 PRINT CHR$(4) „OPEN TEST“
60 PRINT CHR$(4) „READ TEST“
70 INPUT A : INPUT B : INPUT C
80 PRINT CHR$(4) „CLOSE“
```

Dieses Beispiel erscheint auf den ersten Blick umständlicher als die obigen Beispiele. Man muß jedoch berücksichtigen, das das Speichern und Einlesen von Textfiles üblicherweise nicht einer Anhäufung von Einzelvariablen darstellt, z.B.:

```
10 INPUT A1 : INPUT A2 : INPUT A3 : INPUT A4 : INPUT A5 : REM usw.
```

denn dies wäre bei einer großen Anzahl von Variablen völlig unökonomisch. Vielmehr werden Textfiles sinnvollerweise als Arrays gespeichert bzw. eingelesen, z.B.:

```
10 DIM A(1000)
20 PRINT CHR$(4) „OPEN DATEI“
30 PRINT CHR$(4) „READ DATEI“
40 FOR X = 1 TO 1000 : INPUT A(X) : NEXT X
50 PRINT CHR$(4) „CLOSE“
```

Auf diese Weise lassen sich mit wenigen Programmzeilen Tausende von Zahlen oder Strings einlesen.

2.6.5. Semikolon: GET und PRINT CHR$(X)

Semikolons (oder Semikola) bewirken beim PRINT-Befehl die Unterdrückung des Returns sowohl bei der Bildschirmanzeige und Druckerausgabe wie auch bei der Speicherung von Textfiles auf der Diskette. In Applesoft ist es praktisch nicht möglich und auch nicht sinnvoll, Textfiles anzulegen, die keinerlei Returns enthalten. Da vor dem CLOSE-Befehl stets ein Return stehen muß, muß ein mit Applesoft erstellter Textfile zwangsweise ein Return enthalten, z.B.

```
10 PRINT CHR$(4) „OPEN TEST“
20 PRINT CHR$(4) „WRITE TEST“
30 FOR X = 1 TO 10000 : PRINT „AAAAAAAAAA“; : NEXT
40 PRINT : PRINT CHR$(4) „CLOSE“
```

Diese TEST-Datei würde aus 10 mal 10000 = 100.000 A's + einem einzigen Return am File-Ende bestehen. Eine solche fast Return-lose Datei könnte man jedoch mit INPUT-Statements nicht mehr einlesen, da der INPUT-String maximal 239 Zeichen lang sein darf. Mit anderen Worten: Applesoft-Programmierer können grundsätzlich nur Textfiles bewältigen, die in Felder eingeteilt sind, wobei die praktikable Feldlänge in der Regel ca. 230 Zeichen inkl. Return nicht überschreiten sollte.
Ein String als solcher darf bis zu 255 Zeichen lang sein, doch kann ein solcher String nicht mit INPUT eingegeben werden wegen der Obergrenze von 239 Zeichen. Die Definition eines Strings in einer Programmzeile ist sogar auf ca. 230 Zeichen begrenzt, je nachdem, ob die Zeilennummer ein- oder mehrstellig ist:

```
100A$ = ,,1234567890123456789012345678901234567890123456789012345678900123
4567890123456789012345678901234567890123456789012345678901234567890123456789012345
6789012345678901234567890123456789012345678901234567890123456789012345678901234567
890123456789012345678901234567890"
```

Ein einzelner String mit bis 255 Zeichen kann also nur durch Verkettung entstehen:

```
10 A$ = B$ + C$
```

Dateien, die keine (oder fast keine) Returns enthalten, lassen sich von einem Applesoft-Programm mit GET-Befehlen einlesen. Bei Dateien der obigen Größenordnung ist jedoch der GET-Statement völlig undiskutabel, das es eine schiere Ewigkeit dauern würde, bis diese Datei eingelesen wäre. Probieren Sie es selber einmal aus!

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,READ TEST"
30 FOR X = 1 TO 10000
40 FOR Y = 1 TO 10
50 GET X$
60 NEXT Y,Z
70 PRINT : PRINT CHR$(4) ,,CLOSE"
```

Wie lange hat es gedauert?

Übrigens gibt es kein exaktes Gegenstück zum GET-Befehl, denn während der Get-Befehl tatsächlich nur zum Einlesen von je einem einzelnen Zeichen von der Diskette verwendet werden kann, eignet sich der PRINT-Befehl sowohl zum Speichern einzel-

ner Zeichen als auch längerer Zeichenketten (= Strings). Bisweilen ist es erforderlich, daß eine Folge einzelner Zeichen als ASCII-Code in Form von

PRINT CHR$(X);

auf die Diskette gesendet werden müssen, z.B. wenn Binärfiles in Textfiles umgewandelt werden müssen. Ein Beispiel ist hierfür die Umwandlung von Applewriter 1.1-Binärfiles in normale Textfiles, was an dem nachfolgenden Programm demonstriert werden soll:

2.6.5.1. Applewriter 1.1 Binärfile-Konverter

```
100 REM = = = APPLEWRITER 1.1: BINÄR- IN TEXT-FILE = = =
110 HOME : HTAB 14: VTAB 3: INVERSE : PRINT ,,APPLEWRITER 1.1“
120 HTAB 1: VTAB 7: PRINT ,,UMWANDLUNG VON B-FILES IN T-FILES“:
    NORMAL
130 PRINT : PRINT : PRINT ,,START J/N “;
140 GET X$: IF X$ = ,,N“ THEN HOME : END
150 IF X$ < > ,,J“ THEN 140
160 PRINT : PRINT : PRINT ,,LAUFWERK-NR. (1 ODER 2): “;
170 GET X$: IF X$ < > ,,1“ AND X$ < > ,,2“ GOTO 170
180 D = VAL (X$)
190 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D“D
200 INPUT ,,BSAVE-FILE: TEXT.“;Y$
210 PRINT ,,VERTIPPT J/N “;: GET X$: IF X$ < > ,,N“ THEN 190
220 HOME : PRINT : PRINT CHR$ (4),,BLOAD TEXT.“Y$
230 INVERSE : PRINT ,,ABSPEICHERN DES TEXT-FILES“: NORMAL
240 PRINT : PRINT : PRINT ,,LAUFWERK-NR. (1 ODER 2): “;
250 GET X$: IF X$ < > ,,1“ AND X$ < > ,,2“ THEN 250
260 D = VAL (X$)
270 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D“D
280 INPUT ,,TEXT-FILE: “;Y$
290 PRINT ,,VERTIPPT J/N “;: GET X$: IF X$ < > ,,N“ GOTO 270
300 HOME : HTAB 12: INVERSE : PRINT ,,BITTE WARTEN“: NORMAL :
    POKE 34,2: HOME
310 PRINT CHR$ (4),,MON O“
320 PRINT CHR$ (4),,OPEN“Y$: PRINT CHR$ (4),,DELETE“Y$
330 PRINT CHR$ (4),,OPEN“Y$: PRINT CHR$ (4),,WRITE“Y$
340 AD = 6401:Z1 = 32:Z2 = 64:Z3 = 96:Z4 = 128:Z5 = 192:Z6 = 224
```

```
350 A = PEEK (AD):AD = AD + 1
360 IF A = Z3 THEN 420
370 IF A > = Z6 THEN A = A − Z2
380 IF A > = Z5 THEN A = A + Z1
390 IF A < Z1 THEN A = A + Z5
400 IF A < Z2 THEN A = A + Z4
410 PRINT CHR$ (A);: GOTO 350
420 PRINT : PRINT CHR$ (4),,CLOSE": PRINT CHR$ (4),,NOMON O":
    PRINT CHR$ (4),,CATALOG"
430 POKE 34,0
440 INVERSE : PRINT ,,ERNEUT J/N";: NORMAL : PRINT ,, ";
450 GET X$: IF X$ = ,,J" GOTO 110
460 IF X$ < > ,,N" GOTO 450
```

Der PRINT CHR$(A)-Befehl steht bei diesem Programm in Zeile 410. Auch dieses Programm ist ein Beispiel dafür, wie erschreckend lange es dauert, wenn der Zugriff auf einen Textfile auf der Einzelzeichen-Ebene erfolgt.

2.6.6. Komma, Doppelpunkt und Anführungszeichen bei Strings

Da Komma, Doppelpunkt und Anführungszeichen syntaktische Funktionen bei dem INPUT-Befehl übernehmen (technisch am besten beschrieben in dem alten „Basic Programming Reference Manual", S. 66), kann ein auf der Diskette gespeicherter String diese Zeichen nicht alle gleichzeitig enthalten. Es gibt nur zwei Möglichkeiten:

1. Entweder man verzichtet auf die Anführungszeichen (ASCII-Code 34) und kann dann Komma und Doppelpunkt als Bestandteile von Strings auf Textfiles speichern,

2. oder man verzichtet auf Komma und Doppelpunkt und kann dann Anführungszeichen als Bestandteile von Strings auf Textfiles speichern.

Um es klarzustellen: Bei Assembler-Programmen gelten diese Beschränkungen nicht, wohl aber bei reinen Applesoft-Programmen.

Da Komma und Doppelpunkt wichtiger als die doppelten Anführungszeichen (= Gänsefüßchen) sind, zumal ja auch noch die einfachen An- und Abführungszeichen existieren (ASCII-Codes 96 und 39; letzerer zugleich Apostroph), kann man zu dem folgenden, wenig bekannten Trick greifen:

1. Alle auf den WRITE-Befehl folgenden PRINT-Statements werden mit CHR$(34) eingeleitet:

```
10 A$ =,,aaa,aaa:aaa"
20 PRINT CHR$(4) ,,OPEN TEST" : PRINT CHR$(4) ,,WRITE TEST"
30 PRINT CHR$(34) A$
40 PRINT CHR$(4) ,,CLOSE"
```

Ein Semikolon zwischen CHR$(34) und A$, z.B.
30 PRINT CHR$(34) ; A$
dient nur der besseren Lesbarkeit und ist funktional entbehrlich.

2. Bei allen auf den READ-Befehl folgenden INPUT-Statements ist keine besondere Syntax zu beachten, doch muß man wissen, daß bei

```
10 PRINT CHR$(4) ,,OPEN TEST" : PRINT CHR$(4) ,,READ TEST"
20 INPUT A$
30 PRINT CHR$(4) ,,CLOSE"
```

der Variablen A$ nur der reine String aaa,aaa:aaa und nicht etwa Anführungszeichen + aaa,aaa:aaa zugewiesen wird.

Der einzige Nachteil dieses Verfahrens ist der, daß nunmehr jedes Feld auf der Diskette 1 Byte mehr Speicherraum (wegen des Anführungszeichens) beansprucht. Bei langen Strings fällt dies jedoch kaum ins Gewicht. Das nachfolgende Programmbeispiel demonstriert das CHR$(34)-Verfahren:

```
100 REM = = = KOMMA-KOLON-TEST = = =
110 HOME : INVERSE : PRINT ,,KOMMA-KOLON-READ-WRITE-TEST":
    NORMAL
120 PRINT : PRINT : PRINT ,,STARTEN J/N ";: GET X$: IF X$ = ,,N"
    THEN END
130 IF X$ < > ,,J" THEN 110
140 DIM A$(3)
150 X$ = ,,STIEHL,ULRICH:HEIDELBERG"
160 FOR X = 1 TO 3:A$(X) = X$: NEXT
170 HOME : INVERSE : PRINT ,,SO HEISST DER STRING:": NORMAL
180 PRINT : PRINT X$: PRINT : INVERSE : PRINT ,,3 X SPEICHERN:":
    NORMAL : PRINT
190 PRINT CHR$ (4),,MONIO"
```

```
200 PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$ (4),,DELETE
    STRINGTEST": PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$
    (4),,WRITE STRINGTEST"
210 FOR X = 1 TO 3
220 REM = = = ENTSCHEIDENDE ZEILE
230 REM = = = MIT CHR$(34) VOR A$
240 PRINT CHR$ (34) A$(X)
250 NEXT
260 PRINT CHR$ (4),,CLOSE"
270 PRINT : INVERSE : PRINT ,,3 X EINLESEN:": NORMAL : PRINT
280 PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$ (4),,READ
    STRINGTEST"
290 FOR X = 1 TO 3
300 INPUT A$(X)
310 NEXT
320 PRINT CHR$ (4),,CLOSE"
330 PRINT CHR$ (4),,NOMONIO"
340 PRINT : INVERSE : PRINT ,,DAS WURDE EINGELESEN:": NORMAL :
    PRINT
350 FOR X = 1 TO 3: PRINT A$(X): NEXT
```

2.6.7. OPEN-Files

Die Programmierung wird komplizierter, wenn gleichzeitig mehrere Dateien geöffnet sind oder wenn eine Datei nicht geschlossen werden kann, weil laufend oder in Intervallen auf die Datei zugegriffen wird.

Der uns bereits bekannte einfachste Fall liegt dann vor, wenn die PRINT- oder INPUT-Befehle von WRITE-CLOSE- bzw. READ-CLOSE-Befehlen unmittelbar eingeschlossen werden. Was passiert jedoch, wenn zwischendurch Tastatureingaben erwartet oder Hinweise am Bildschirm angezeigt werden sollen? In diesem Fall muß der READ- oder WRITE-Befehl vorübergehend inaktiviert werden, damit nicht etwa der Bildschirm-Hinweis plötzlich auf der Diskette gespeichert wird. Die vorübergehende Inaktivierung von READ-WRITE-Befehlen ist denkbar einfach durch einen reinen PRINT CHR$(4)-Befehl zu realisieren. (Übrigens führt jeder ANDERE DOS-Befehl, z.B. PRINT CHR$(4) ,,CATALOG", wegen des Ctrl-D zu einer Inaktivierung des momentanen Read- oder Write-Zustandes.)

Beispiel 1:

```
10 PRINT CHR$(4) „OPEN DATEI"
20 PRINT CHR$(4) : REM INAKTIVERT DOS
30 PRINT „STRING EINGEBEN!"
40 INPUT X$
50 PRINT CHR$(4) „WRITE DATEI"
60 PRINT X$ : GOTO 20
```

Hier würde eine fehlende Zeile 20 zur Folge haben, daß neben der Variablen X$ auf der Diskette ungewollt jeweils der String „STRING EINGEBEN!" mit abgespeichert werden würde.

Beispiel 2:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 FOR X = 1 TO 100 : PRINT X : NEXT
40 PRINT CHR$(4) „CLOSE"
50 PRINT CHR$(4) „OPEN TEST"
60 PRINT CHR$(4) : REM INAKTIVIERT DOS
70 PRINT „ZAHL EINLESEN J/N ": INPUT X$
80 IF X$ = „N" THEN PRINT CHR$(4) „CLOSE" : END
90 PRINT CHR$(4) „READ TEST" : INPUT Z : PRINT Z : GOTO 60
```

Hier würde eine fehlende Zeile 60 zur Folge haben, daß bei Zeile 70 mit INPUT X$ die nächste Zahl automatisch von der Diskette käme, ohne daß man eine Chance hätte, durch Tastatureingabe von „N" das Programm abzubrechen.

Die vorübergehende DOS-Inaktivierung, die übrigens nicht mit dem DOS-„Abhängen" durch PR#0 : IN#0 zu verwechseln ist, hat übrigens den Vorteil, daß eventuelle DOS- oder Applesoft-Fehlermeldungen nicht im Falle des WRITE-Aktivzustandes aus Versehen auf der Diskette gespeichert werden.

Wenn mehrere Dateien gleichzeitig geöffnet sind, kommen oft noch die Parameter Slot, Drive und Volume zum Tragen. Beispiel:

```
10 PRINT „A-DATEI:" : PRINT
20 INPUT „DATEINAME, SLOT, DRIVE, VOLUME: "; DA$, SA, DA, VA
30 PRINT „B-DATEI:" : PRINT
```

```
40 INPUT „DATEINAME, SLOT, DRIVE, VOLUME: "; DB$, SB, DB, VB
50 PRINT CHR$(4) „OPEN"DA$ „,S"SA „,D"DA „,V"VA
60 READ CHR$(4) „READ"DA$
70 PRINT CHR$(4) „OPEN"DB$ „,S"SB „,D"DB „,V"VB
80 READ CHR$(4) „READ"DB$
```

Ein im meinen Augen lästiger Fehler des DOS-Betriebssystems ist der, daß die READ- und WRITE-Befehle im Gegensatz zum OPEN-Befehl keine Parameter zulassen. Deshalb ist es ohne Poke-Befehle nicht möglich, auf mehrere, sich in verschiedenen Drives und/oder Slots befindliche Dateien gleichzeitig zuzugreifen, ohne die jeweilige Datei vorher zu schließen.

Da Apple-Besitzer in der Regel nur über 2 Laufwerke verfügen, stellt sich das Problem des Slot-Wechsels selten, und Volume-Nummern werden ohnehin kaum benutzt. Dagegen ist die Notwendigkeit des Drive-Wechsels offenkundig. Angenommen, man wollte zwei in sich sortierte Textfiles, die so groß sind, daß sie nicht komplett in den RAM-Speicher passen, von Drive 1 nach Drive 2 mischen (= Merge- oder Mischvorgang = Vereinigung zweier in sich sortierter Teildateien zu einer neuen ebenfalls in sich sortierten großen Gesamtdatei). Zu diesem Zweck müssen vorübergehend alle drei Dateien offen sein.

Drive-Wechsel:

Mit POKE -21912,1 kann man Drive 1 als aktives Drive deklarieren.
Mit POKE -21912,2 kann man Drive 2 als aktives Drive deklarieren.

Slot-Wechsel:

Mit POKE -21910,4 kann man Slot 4 als aktiven Slot deklarieren.
Mit POKE -21910,6 kann man Slot 6 als aktiven Slot deklarieren, usw.

Diese Pokes gelten für 48K-DOS, also nicht für den Fall, daß sich DOS in der Language Card befindet.

2.6.7.1. „Schein-Mischen" (Demo-Programm)

```
100  PRINT CHR$(4) „OPEN A1, D1"
110  PRINT CHR$(4) „WRITE A1"
120  FOR X = 1 TO 1000
```

```
130 PRINT „AAAAAAAAAAAAAAA“ : NEXT
140 PRINT CHR$(4) „CLOSE“
150 PRINT CHR$(4) „OPEN A2“
160 PRINT CHR$(4) „WRITE A2“
170 FOR X = 1 TO 1000
180 PRINT „AAAAAAAAAAAAAAA“ : NEXT
190 PRINT CHR$(4) „CLOSE“
200 REM SCHEIN-MERGEN
210 DIM A1$ (100), A2$ (100)
220 PRINT CHR$(4) „OPEN A1“
230 PRINT CHR$(4) „OPEN A2“
240 POKE -21912,2 : REM DRIVE 2
250 PRINT CHR$(4) „OPEN M“
260 FOR X = 1 TO 10
270 POKE -21912,1 : REM DRIVE 1
280 PRINT CHR$(4) „READ A1“
290 FOR Y = 1 TO 100 : INPUT A1$(X) : NEXT
300 PRINT CHR$(4) „READ A2“
310 FOR Y = 1 TO 100 : INPUT A2$(X) : NEXT
320 POKE -21912,2 : PRINT CHR$(4) „WRITE M“: REM DRIVE 2
330 FOR Y = 1 TO 100 : PRINT A1$(X) : PRINT A2$(X) : NEXT
340 NEXT X
350 PRINT CHR$(4) „CLOSE“
```

Ohne die angegebenen Poke-Befehle wäre es bei den geöffneten Dateien A1, A2 und M nicht möglich gewesen, den Drive-Wechsel aus dem Programm heraus vorzunehmen.

2.6.7.2. „Echtes Mischen“ (Registerprogramm)

Das nachfolgende komplizierte Programm ist ein Teil-Modul zu meinem Registerprogramm, das vor allem für Register am Ende von Büchern (Stichwort + Seitenzahl) Verwendung findet. Die Dateistruktur ist folgende:

1. Feld: Anzahl der Stringpaare (als 6stelliger String)
2. Feld: CHR$(34) + 1. Stichwort
3. Feld: CHR$(34) + 1. Seitenzahl
4. Feld: CHR$(34) + 2. Stichwort
5. Feld: CHR$(34) + 2. Seitenzahl

usw. bis zu den 2 letzten Feldern, die ,,ULI", ,,ULI" als programmtechnische Endmarker haben.

Beispiel:

000004
,,Computer
,,10
,,DOS 3.3
,,30
,,Apple IIe
,,20
,,formatieren
,,70
,,ULI
,,ULI

Es sind also sowohl die Stichwörter als auch die Seitenzahlen als Strings gespeichert, da das Programmpaket auch für andere Register, z.B. Glossare, verwendet werden kann. Es werden automatisch bis zu 20 in sich bereits sortierte, beliebig große Dateien paarweise von Drive 1 eingelesen, gemischt und als neue Misch-Datei auf Drive 2 abgespeichert. Das paarweise Mischen muß solange vollzogen werden, bis nur noch eine einzige ,,Monster"-Enddatei übrigbleibt, die übrigens mehr als 300.000 Zeichen Umfang haben kann. Da das Mischen ein langwieriger Prozeß ist, ist das Applesoft-Programm mit dem TASC compiliert (mit Origin $0806). Außerdem wird Diversi-DOS benutzt, das in die Language Card geschoben wird. Aber auch dies reicht noch nicht aus, da Sortieren und Mischen nach DUDEN erfolgen soll, dessen Alphabetisierungsrichtlinien sich bekanntlich nicht mit der ASCII-Code-Reihenfolge decken. Deshalb erfolgt die Umcodierung durch ein Assemblerprogramm, das die Strings auch nicht mehr mit dem INPUT-Befehl, sondern mit dem Monitor-GETLN-Befehl ($FD6A) einliest, umcodiert und dann in der ,,Schein-String-Zeile" 150 ablegt, so daß das compilierte Applesoft-Programm diesen String übernehmen kann. Auf diese Weise — Diversi-DOS, DOS in der LC, Compilierung, Assemblerroutinen — ist das Registermischprogramm etwa 10-20 mal schneller, als ein entsprechendes ,,normales" Applesoft-Programm unter DOS 3.3 wäre. Was allein 10mal schneller bedeutet, wird einem klar, wenn man statt 1 Minute z.B. 10 Minuten am Bildschirm wartet.

Wenn DOS oder Diversi-DOS in der Language Card liegt, ist das Umschalten von Drive 1 nach Drive 2 bei geöffneten Textfiles besonders kompliziert und kann wegen

des erforderlichen ,,Bank-Selecting" nur durch ein kleines Assemblerprogramm realisiert werden. Das Umschalten geschieht in dem nachstehenden Programm durch die Befehle CALL 826 (= Drive 1) und CALL 830 (= Drive 2). Ferner kommt in dem Programm CALL 849, das das Sortierwort unmodifiziert mit GETLN einliest, sowie CALL 875 vor, das die Umcodierung vornimmt.

```
100  REM = = = REGISTERMISCHER = = =
110  REM !INTEGER*
120  REM !DEFCOMMONA,D
130  REM !DEFCOMMOND$(20),D$
140  GOTO 190
150  M$ = ,,": M$ = LEFT$ (,,?--------------------------------------------------------------
     --------------",MM): RETURN
160  CALL 849:MM = PEEK (255): GOSUB 150:M0$ = M$
170  CALL 875:MM = PEEK (255): GOSUB 150:C0$ = M$
180  INPUT N0$: RETURN
190  POKE 846,10: POKE 847,24: REM $180A:?---
200  ONERR GOTO 220
210  GOTO 280
220  PRINT CHR$ (4): PRINT CHR$ (4),,CLOSE": IF PEEK (222) = 9 THEN
     PRINT CHR$ (4),,DELETE"A0$
230  TEXT : HOME : PRINT ,,FEHLER-NR. " PEEK (222)
240  PRINT CHR$ (7)
250  PRINT ,,W = WEITER ";
260  GET D$: IF D$ < > ,,W" THEN 260
270  REM MENUE
280  HOME : INVERSE : PRINT ,, REGISTERMISCHER ": NORMAL
290  PRINT : PRINT ,,(MAX.10 SORTIERTE DATEIPAARE MISCHBAR)"
300  PRINT : PRINT ,,(AUSGANGSDATEIEN,D1 -> ZIELDATEIEN,D2)"
310  PRINT : PRINT ,,1 MISCHEN VON DATEIPAAREN": PRINT : PRINT
     ,,5 UEBERTRAGEN EINER RESTDATEI": PRINT : PRINT ,,0
     HAUPTMENUE": PRINT
320  INVERSE : PRINT ,,VON ULRICH STIEHL - VERSION V. 01.02.84":
     NORMAL : PRINT
330  GET D$: ON D$ = ,,1" GOTO 350: ON D$ = ,,5" GOTO 940: ON D$ =
     ,,0" GOTO 540: GOTO 330
340  REM DATEIENEINGABE
350  CLEAR : DIM D$(20)
360  HOME : PRINT : PRINT CHR$ (4),,CATALOG,D1": INVERSE :
     PRINT ,,RETURN ALLEIN = EINGABEENDE": NORMAL
```

```
370 D = 0
380 D = D + 1: INPUT ,,A-DATEI:";D$(D)
390 D = D + 1: INPUT ,,B-DATEI:";D$(D)
400 IF D$(D - 1) = ,," AND D$(D) = ,," AND D = 2 THEN 280
410 IF D$(D - 1) = ,," AND D$(D) = ,," THEN D = D - 2: GOTO 440
420 IF D$(D - 1) = ,," OR D$(D) = ,," THEN 360
430 IF D < 19 THEN 380
440 PRINT ,,RICHTIG J/N ";
450 GET D$: ON D$ = ,,N" GOTO 350: IF D$ < > ,,J" GOTO 450
460 GOSUB 530: PRINT : FOR X = 1 TO D: PRINT CHR$
    (4),,UNLOCK"D$(X),,,D1": NEXT
470 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D2": INPUT ,,ZIEL-
    DATEINAME:";D$: ON D$ = ,," GOTO 470: PRINT ,,RICHTIG J/N ";
480 GET X$: ON X$ = ,,N" GOTO 470: IF X$ < > ,,J" GOTO 480
490 REM MISCHSTART
500 A = - 1: HOME
510 A = A + 2: IF A + 1 < = D THEN 570
520 HOME : PRINT ,,MISCHEN FERTIG !": GOTO 240
530 HOME : HTAB 14: VTAB 10: INVERSE : PRINT ,,BITTE WARTEN":
    NORMAL : RETURN
540 HOME : INVERSE : PRINT ,,PROGRAMMDISKETTE IN DRIVE 1":
    NORMAL : PRINT CHR$ (7): INPUT ,,HAUPTMENUE EINLESEN
    JA/N ";D$: IF D$ < > ,,JA" THEN 280
550 GOSUB 530: PRINT : PRINT CHR$ (4),,BRUN
    REGISTERSTARTER.OBJ,D1"
560 REM = = = MISCHEN = = =
570 CLEAR :L = 150: DIM T$(150),I$(150)
580 GOSUB 530: HTAB 1: VTAB 1: PRINT D$(A): PRINT D$(A + 1)
590 REM ERSTER STRING
600 A1$ = D$(A):A2$ = D$(A + 1)
610 A0$ = D$ + ,,.M" + STR$ ((A + 1) / 2)
620 PRINT : PRINT CHR$ (4),,OPEN"A1$,,,D1": PRINT CHR$
    (4),,READ"A1$: INPUT X$:X = VAL (X$)
630 PRINT CHR$ (4),,OPEN"A2$: PRINT CHR$ (4),,READ"A2$: INPUT
    Y$:Y = VAL (Y$)
640 CALL 830: REM LAUFWERK 2
650 PRINT CHR$ (4),,OPEN"A0$: PRINT CHR$ (4),,WRITE"A0$:X = X
    + Y:X$ = STR$ (X):Y$ = ,,000000":X$ = LEFT$ (Y$,6 - LEN (X$)) +
    X$: PRINT X$
660 CALL 826: REM LAUFWERK 1
```

```
670 PRINT CHR$ (4),,READ"A1$: GOSUB 160:M1$ = M0$:C1$ = C0$:N1$
    = N0$
680 PRINT CHR$ (4),,READ"A2$: GOSUB 160:M2$ = M0$:C2$ = C0$:N2$
    = N0$
690 B = 1
700 IF C1$ > C2$ GOTO 780
710 REM = = = = C1$<=C2$ = = = =
720 T$(B) = M1$:I$(B) = N1$
730 B = B + 1: IF B > L THEN GOSUB 900
740 PRINT CHR$ (4),,READ"A1$: GOSUB 160:M1$ = M0$:C1$ = C0$:N1$
    = N0$
750 IF M1$ = ,,ULI" AND N1$ = ,,ULI" THEN PRINT CHR$
    (4),,CLOSE"A1$:A$ = A2$:T$(B) = M2$:I$(B) = N2$: GOTO 840
760 GOTO 700
770 REM = = = = C1$>C2$ = = = =
780 T$(B) = M2$:I$(B) = N2$
790 B = B + 1: IF B > L THEN GOSUB 900
800 PRINT CHR$ (4),,READ"A2$: GOSUB 160:M2$ = M0$:C2$ = C0$:N2$
    = N0$
810 IF M2$ = ,,ULI" AND N2$ = ,,ULI" THEN PRINT CHR$
    (4),,CLOSE"A2$:A$ = A1$:T$(B) = M1$:I$(B) = N1$: GOTO 840
820 GOTO 700
830 REM A$=RESTDATEI
840 B = B + 1: IF B > L THEN GOSUB 900
850 PRINT CHR$ (4),,READ"A$: INPUT T$(B): INPUT I$(B)
860 IF T$(B) < > ,,ULI" AND I$(B) < > ,,ULI" GOTO 840
870 L = B: GOSUB 900
880 PRINT CHR$ (4),,CLOSE": PRINT CHR$ (4),,DELETE"A1$,,,D1":
    PRINT CHR$ (4),,DELETE"A2$: GOTO 510
890 REM SPEICHERN EINES L-BLOCKS
900 CALL 830
910 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO L: PRINT CHR$
    (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
920 FOR X = 1 TO L:T$(X) = ,,":I$(X) = ,,": NEXT : CALL 826:B = 1:
    RETURN
930 REM ÜBERTRAGEN
940 CLEAR :L = 150: DIM T$(150),I$(150): HOME : INVERSE : PRINT
    ,,UEBERTRAGEN EINER DATEI ": PRINT ,,VON DRIVE 1 NACH
    DRIVE 2": NORMAL
950 PRINT : PRINT ,,1 = ZURUECK": PRINT : PRINT ,,0 = UEBER-
```

```
     TRAGEN": PRINT
960  GET D$: ON D$ = ,,1" GOTO 280: IF D$ < > ,,0" GOTO 960
970  HOME : PRINT : PRINT CHR$ (4),,CATALOG,D1": INPUT ,,ALTER
     DATEINAME: ";X$: ON X$ = ,," GOTO 940: PRINT
     ,,RICHTIG J/N ";
980  GET Y$: ON Y$ = ,,N" GOTO 970: IF Y$ < > ,,J" GOTO 980
990  HOME : PRINT : PRINT CHR$ (4),,CATALOG,D2": INPUT ,,NEUER
     DATEINAME: ";A0$: ON A0$ = X$ GOTO 940: PRINT
     ,,RICHTIG J/N ";
1000 GET Y$: ON Y$ = ,,N" GOTO 990: IF Y$ < > ,,J" GOTO 1000
1010 GOSUB 530: PRINT : PRINT CHR$ (4),,OPEN"X$,,,D1": PRINT CHR$
     (4),,READ"X$: INPUT Y$:B = VAL (Y$)
1020 PRINT CHR$ (4),,OPEN"A0$,,,D2": PRINT CHR$ (4),,DELETE"A0$:
     PRINT CHR$ (4),,OPEN"A0$: PRINT CHR$ (4),,WRITE"A0$:
     PRINT Y$
1030 IF B > L THEN 1110
1040 REM B< =L;B+1=,,ULI"
1050 CALL 826
1060 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO B + 1: INPUT T$(X):
     INPUT I$(X): NEXT
1070 CALL 830
1080 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO B + 1: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT : PRINT CHR$ (4),,CLOSE"
1090 PRINT CHR$ (4),,DELETE"X$,,,D1": GOTO 280
1100 REM B>L;D=REST
1110 A = INT (B / L):D = B + 1 − A · L
1120 FOR Z = 1 TO A
1130 CALL 826
1140 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO L: INPUT T$(X): INPUT
     I$(X): NEXT
1150 CALL 830
1160 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO L: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
1170 NEXT Z
1180 CALL 826
1190 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO D: INPUT T$(X): INPUT
     I$(X): NEXT
1200 CALL 830
1210 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO D: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
```

1220 PRINT CHR$ (4),,CLOSE“: PRINT CHR$ (4),,DELETE“X$,,,D1“:
GOTO 280

```
:ASM
                          ORG   826
                  *
                  ********************************
                  *
                  HC08B    EQU   $C08B
                  HC081    EQU   $C081
                  HC083    EQU   $C083
                  HEA68    EQU   $EA68
                  HC082    EQU   $C082
                  *
                  *
                  ********************************
                  *
033A: A9 01       DRIVE1   LDA   #$01        ;826
033C: D0 02                BNE   DRIVE
033E: A9 02       DRIVE2   LDA   #$02        ;830
0340: AE 83 C0    DRIVE    LDX   HC083
0343: AE 83 C0             LDX   HC083
0346: 8D 68 EA             STA   HEA68
0349: AD 82 C0             LDA   HC082
034C: 60                   RTS
                  *
                  ********************************
                  *
                  STRLEN   EQU   255         ;$FF
                  PUFFER   EQU   $201        ;NACH "
                  GETLN    EQU   $FD6A
                  *
034D: 9D FF FF    POKER    STA   $FFFF,X     ;846/847
0350: 60                   RTS
                  *
                  * DARF KEIN NULLSTRING SEIN,
                  * D.H. NUR RETURN!!!
                  *
0351: 20 6A FD    GETLN1   JSR   GETLN       ;849
0354: A2 00                LDX   #0
0356: BD 01 02    MOVESTR1 LDA   PUFFER,X
0359: 29 7F                AND   #$7F
035B: 9D 01 02             STA   PUFFER,X
035E: C9 0D                CMP   #$0D        ;RETURN
0360: F0 06                BEQ   MOVEEXIT
0362: 20 4D 03             JSR   POKER
0365: E8                   INX
0366: D0 EE                BNE   MOVESTR1
0368: 86 FF       MOVEEXIT STX   STRLEN
036A: 60                   RTS
                  *
                  ********************************
```

```
                       *
036B: A2 00            GETLN2   LDX   #0          ;875
036D: A0 00                     LDY   #0
                       *
036F: B9 01 02         MOVESTR2 LDA   PUFFER,Y
0372: C9 0D                     CMP   #$0D        ;RETURN
0374: F0 4C                     BEQ   EXIT1
                       *
0376: C9 7E                     CMP   #$7E        ;ß
0378: F0 2E                     BEQ   ESZETT
037A: C9 7D                     CMP   #$7D        ;ü
037C: F0 2E                     BEQ   UBUCHST
037E: C9 7C                     CMP   #$7C        ;ö
0380: F0 2E                     BEQ   OBUCHST
0382: C9 7B                     CMP   #$7B        ,ä
0384: F0 2E                     BEQ   ABUCHST
                       *
0386: C9 5D                     CMP   #$5D        ;Ü
0388: F0 22                     BEQ   UBUCHST
038A: C9 5C                     CMP   #$5C        ,Ö
038C: F0 22                     BEQ   OBUCHST
038E: C9 5B                     CMP   #$5B        ;Ä
0390: F0 22                     BEQ   ABUCHST
                       *
0392: C9 61                     CMP   #$61        ;a-z
0394: B0 22                     BCS   GRBUCHST
0396: C9 5E                     CMP   #$5E        ;^_`
0398: B0 25                     BCS   NICHTS
039A: C9 41                     CMP   #$41        ;A-Z
039C: B0 1D                     BCS   MOVESTR3
                       *
039E: C9 30                     CMP   #'0'
03A0: 90 1D                     BCC   NICHTS      ;<O
03A2: C9 3A                     CMP   #':'
03A4: 90 15                     BCC   MOVESTR3    ;ZIFFER
03A6: B0 17                     BCS   NICHTS
                       *
03A8: A9 53            ESZETT   LDA   #'S'        ;ß>S
03AA: D0 0F                     BNE   MOVESTR3
03AC: A9 55            UBUCHST  LDA   #'U'        ;ä+A>A
03AE: D0 0B                     BNE   MOVESTR3
03B0: A9 4F            OBUCHST  LDA   #'O'        ;ö+Ö>O
03B2: D0 07                     BNE   MOVESTR3
03B4: A9 41            ABUCHST  LDA   #'A'        ;ä+A>A
03B6: D0 03                     BNE   MOVESTR3
03B8: 38               GRBUCHST SEC               ;KL>GR
03B9: E9 20                     SBC   #$20
                       *
03BB: 20 4D 03         MOVESTR3 JSR   POKER
03BE: E8                        INX               ;LENGTH
03BF: C8               NICHTS   INY               ;PUFFCNT
03C0: D0 AD                     BNE   MOVESTR2
03C2: E0 00            EXIT1    CPX   #0
03C4: D0 06                     BNE   EXIT2
```

```
03C6: A9 2F      103          LDA  #'/'        ;LEER
03C8: 20 4D 03   104          JSR  POKER
03CB: E8         105          INX
03CC: 86 FF      106  EXIT2   STX  STRLEN
03CE: 60         107          RTS
```

Näheres über die GETLN-Routine steht im Teil II, Kapitel 4. Insbesondere beachte man, daß GETLN im Gegensatz zu INPUT das Anführungszeichen mit einliest.

2.6.8. Sequentielle und Random-Files

Sequentielle File sind Textfiles mit variablen Feldlängen und indirektem Zugriff, Random-Files sind Textfiles mit konstanten Feldlängen und direktem Zugriff. (Random bzw. genauer Random Access bedeutet wahlfreier = direkter Zugriff auf den Diskettensektor.)

Die bisher behandelten Beispiele waren — auch wenn dies nicht ausdrücklich erwähnt wurde — sogenannte sequentielle Textfiles. Ein typischer sequentieller Textfile wie z.B. bei dem oben beschriebenen Registerprogramm setzt sich aus einer beliebigen Anzahl unterschiedlich — die Betonung liegt auf unterschiedlich! — langer Felder zusammen, die wie eine Sequenz (= Folge) aneinandergereiht und lediglich durch Returns getrennt sind. DOS hängt an das Ende eines Textfiles automatisch ein Ctrl-0 als sozusagen DOS-internen Endmarker. Ein sequentielle Dateien verarbeitendes Applesoft-Programm sollte darüber hinaus eigene „Buchführung" machen, damit die „END OF FILE"-Fehlermeldung vermieden wird. Bei dem obigen Registerprogramm gibt es

1. als ersten String der Datei einen Anfangsmarker als normierte, 6stellige, rechtsbündig ausgeschlossene Zahl, die die Anzahl der Doppelstrings (Stichwort + Seitenzahl) beinhaltet, sowie
2. als letzte 2 Strings der Datei zwei Endmarker („ULI", „ULI"), damit man niemals beim Ctrl-0 ankommt

Warum eine normierte 6stellige Zahl als String, mag man sich fragen. Da sequentielle Dateien Felder mit unterschiedlicher Länge haben und Zahlen selbst eine unterschiedliche Anzahl von Speicherstellen auf der Diskette einnehmen können (1, 10, 100, 1000 usw.), wäre es bei einer sequentiellen Datei nachträglich nicht mehr mög-

lich, den Anfangsmarker korrekt durch einen neuen Zahlenwert (z.B. bei gemischten Dateien) zu überschreiben.

Sequentielle Dateien sind Textfiles mit indirektem Zugriff analog zu Dateien, die auf Datenkassetten, Lochstreifen usw. gespeichert sind. Wenn man z.B. das 150. und nur dieses 150. Feld einer sequentiellen Datei einlesen möchte, so bleibt einem nichts anderes übrig, als alle vorangehenden 149 Felder mit einzulesen, da DOS keine Möglichkeit hat, auf sozusagen rechnerischem Wege zu ermitteln, auf welchem Sektor und auf welcher Spur sich exakt das 150. Feld befindet. Man könnte zwar auch hierfür eine „Buchhaltung" einführen, doch würde diese ihrerseits besonders bei kurzen Feldern relativ viel Platz auf der Diskette beanspruchen, von dem Zeitaufwand für die Anlage einer solchen „Buchführung" ganz zu schweigen. Aber selbst wenn all dies kein Hindernis wäre, dann wäre diese „Buchführung" trotzdem ein (für Diskettenlaufwerke) nutzloses Unterfangen, denn nachträglich läßt sich bei einem sequentiellen File kein Feld mehr erweitern oder kürzen, ohne daß die Datei insgesamt neu abgespeichert wird. Dies ist zwar bei Festplattenlaufwerken mit hoher Zugriffszeit, nicht jedoch bei normalen Diskettenlaufwerken mit vergleichsweise sehr geringer Zugriffszeit, akzeptabel.

Der große Vorteil sequentieller Dateien beruht auf der Kompaktheit der Datenspeicherung, da gewissernmaßen jede Speicherstelle auf der Diskette voll ausgenutzt wird. Hinsichtlich des Verwendungszwecks eignen sich sequentielle Files für solche Dateien, bei denen man nicht auf das EINZELNE Feld zugreifen muß, sondern wo die GESAMTHEIT der Felder im Vordergrund steht. Ein typisches Beispiel für sequentielle Dateiverwaltung ist damit ein Registerprogramm, weil hier die Stichwörter wahllos (z.B. nach dem Buchumbruch) eingegeben werden und die fertig sortierte Enddatei mit der Übertragung in die Photosetzmaschine ihren Zweck erfüllt hat.

Wenn dagegen Datensätze, z.B. Kundenadressen usw., ständig gepflegt werden müssen (durch Neueingabe, Änderung und Löschung), sollte man anstelle sequentieller Dateien Random-Files anlegen. Random-Dateien sind Dateien mit direktem Zugriff, da die Records oder Datensätze eine vordefinierte Länge haben. Nehmen wir an, eine Random-Datei bestehe aus 501 Records — numeriert von 0-500 — mit je 128 Zeichen Länge, dann ist es für DOS ein leichtes auszurechnen, auf welchem Sektor und auf welcher Spur sich z.B. der 150. Record befindet. Da ein Sektor 256 Bytes umfaßt, passen in diesem Fall 2 Records auf einen Datensektor. Im 1. Datensektor befinden sich die Records 0 und 1, im 2. Datensektor die Records 2 und 3 usw. Somit befindet sich der 150. Record in der ersten Hälfte des 75. Datensektors.

Random-Files werden geöffnet mit einer Längenabgabe (L = Recordlänge) als zu-

sätzlicher Parameter zum OPEN-Befehl. Ferner muß bei jedem READ- oder WRITE-Befehl die Record-Nummer (R) spezifiziert werden. Beispiel:

```
10 PRINT CHR$(4) ,,OPEN TELEFONLISTE,L50"
20 R = 1
30 PRINT CHR$(4)
40 INPUT ,,NAME + TEL. "; T$
50 PRINT CHR$(4) ,,WRITE TELEFONLISTE,R"R
60 PRINT X$ : R = R + 1 : GOTO 30
```

Die Record-Nummern R beginnen immer bei 0 und gehen maximal bis 32767. Die Record-Länge L muß im Bereich 1-32767 liegen. Normalerweise benutzt man den Record 0 nicht für eigentliche Daten, sondern z.B. für die Speicherung der Zahl der bisher belegten Records. Ein einzelner Record besteht meist aus mehreren Feldern, z.B. Vorname, Zuname, Straße usw. In diesem Sinne ist Record das Fremdwort für Datensatz. Speicher- und programmtechnisch lassen sich 2 Typen von Datensätzen unterscheiden:

Typ 1: Felder durch Returns abgegrenzt (Auszug)

```
 10  PRINT CHR$(4) ,,OPEN ADRESSEN,L132" : REM 128 + 4 RETURNS
 20  R = 1
 30  PRINT CHR$(4)
 40  INPUT ,,VORNAME: "; V$
 50  INPUT ,,ZUNAME: "; Z$
 60  INPUT ,,STRASSE: "; S$
 70  INPUT ,,PLZ und ORT: "; P$
 80  F = LEN (V$) + LEN (Z$) + LEN (S$) + LEN (P$)
 90  IF F > 128 THEN PRINT ,,RECORD ZU GROSS" : GOTO 30
100  PRINT CHR$(4) ,,WRITE ADRESSEN,R"R
110  PRINT V$ : PRINT Z$ : PRINT S$ : PRINT P$ : R = R + 1 : GOTO 30
```

Das Einlesen würde beim Typ 1 so aussehen (Auszug):

```
500  PRINT CHR$(4) ,,READ ADRESSEN,R"R
510  INPUT V$ : INPUT Z$ : INPUT S$ : INPUT P$
520  PRINT CHR$(4)
530  PRINT V$ : PRINT Z$ : PRINT S$ : PRINT P$
540  RETURN
```

Typ 2: Record durch Return abgegrenzt (Auszug)

```
 10 PRINT CHR$(4) „OPEN ADRESSEN,L129“ : REM 128 + 1 RETURN
 20 R = 1
 30 PRINT CHR$(4)
 40 INPUT „VORNAME: “; V$ : V = LEN (V$) : IF V > 32 GOTO 40
 50 INPUT „ZUNAME: “; Z$ : Z = LEN (Z$) : IF Z > 32 GOTO 50
 60 INPUT „STRASSE: “; S$ : S = LEN (S$) : IF S > 32 GOTO 60
 70 INPUT „PLZ und ORT: “; P$ : P = LEN (P$) : IF P > 32 GOTO 70
 80 PRINT CHR$(4) „WRITE ADRESSEN,R“R
 90 F = V : PRINT V$; : GOSUB 140
100 F = Z : PRINT Z$; : GOSUB 140
110 F = S : PRINT S$; : GOSUB 140
120 F = P : PRINT P$; : GOSUB 140
130 PRINT : R = R + 1 : GOTO 30
140 D = 32 − F : IF D = 0 THEN 180
150 FOR X = 1 TO D
160 PRINT CHR$(32); : REM LEERTASTE
170 NEXT X
180 RETURN
```

Das Einlesen würde beim Typ 2 so aussehen (Auszug):

```
500 PRINT CHR$(4) „READ ADRESSEN,R“R
510 INPUT A$ : PRINT CHR$(4)
520 PRINT LEFT$ (A$, 32)
530 PRINT MID$ (A$, 33, 32)
540 PRINT MID$ (A$, 65, 32)
550 PRINT RIGHT$ (A$, 32)
560 RETURN
```

Beim Typ 1 ist jedes einzelne Feld durch ein Return vom nachfolgenden abgegrenzt, während beim Typ 2 lediglich das letzte Feld des Records, also der Record selbst, ein Return erhält. Der Typ 1 ist trotz der zusätzlichen Returns speicherökonomischer, da keine unnötigen Leertasten am Feldende verschwendet werden. Für Bildschirmmasken sind derartig variable Feldlängen jedoch ungeeignet. Ferner kann bei solchen Feldern kein formatierter, tabellarischer Ausdruck mehr zustande kommen, denn wie will man eine mehrspaltige Tabelle einrichten, wenn z.B. die Straße mal 35 Zeichen und mal 15 lang ist. Hinzu kommt, daß die Datentypistin sozusagen erst am Schluß weiß, ob sie kürzen muß oder nicht, da ja die tatsächliche Recordlänge erst mit der Eingabe des letzten Feldes vorliegt.

Der nachfolgende kleine Random-Test zeigt, daß Random-Files natürlich nicht nur „at random", d.h. zufällig oder wahlfrei, sondern auch sequentiell gelesen werden können. Allerdings ist jeweils der R-Parameter erforderlich.

```
100 REM = = = RANDOM-TEST = = =
110 PRINT CHR$ (4),,OPENRANDOM,L200"
120 FOR X = 1 TO 200
130 PRINT CHR$ (4),,WRITERANDOM,R"X
140 PRINT X · 1.2345
150 NEXT X
160 PRINT CHR$ (4),,CLOSE"
170 INPUT ,,(1-200)?"; A
180 A = A · 1.2345
190 PRINT CHR$ (4),,OPENRANDOM,L200"
200 FOR X = 1 TO 200
210 PRINT CHR$ (4),,READRANDOM,R"X
220 INPUT B
230 IF A = B THEN PRINT A,B: GOTO 270
240 NEXT X
250 PRINT CHR$ (4),,CLOSE"
260 END
270 PRINT ,,ENDE"
280 PRINT CHR$ (4),,CLOSE"
```

2.6.8.1 Vorformatierte Random-Files

Es gibt jedoch auch die Möglichkeit, Random-Files anzulegen, die zugleich sequentielle Dateien sind. Bei dem obigen Random-Test wird jeweils nur eine einzige Zahl in einem 200stelligen Record abgelegt. In Erweiterung unserer Erläuterungen zur Struktur von Textfiles gilt für Random-Files, daß nicht nur nach dem allerletzten Record, sondern auch zwischen den einzelnen Records NUL-Codes (Ctrl-0) abgespeichert sein können. Zur Verdeutlichung folgendes Extrembeispiel:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,DELETE TEST"
30 PRINT CHR$(4) ,,OPEN TEST,L"100
40 PRINT CHR$(4) ,,WRITE TEST,R"1000
50 PRINT ,,1"
60 PRINT CHR$(4) ,,CLOSE"
```

Dieses Programm würde einen Random-File erzeugen, der (später) fast die ganze Diskette einnehmen würde, obgleich zunächst nur eine einzige „1“ abgespeichert wäre. Wäre die Datendiskette zuvor initialisiert worden, dann hätte die TEST-Datei folgende Struktur:

100 mal 1000 (0-999) Ctrl-0's + „1“ + Return + 98 Ctrl-0's

Hätte es sich dagegen um eine alte Diskette gehandelt, auf der zahlreiche Files bereits gelöscht worden wären, dann würde die TEST-Datei anstelle „sauberer“ Ctrl-0's „wirren Schrott“ in Form von Resten alter Programme usw. enthalten.

Besser ist es, wenn man Random-Dateien quasi vorformatiert in dem Sinne, daß man die ganze Datei z.B. mit Leertasten, Nullen o.ä. beschreibt mit der Folge, daß solch eine Random-Datei dann auch wie eine sequentielle Datei behandelt werden kann, weil das Ctrl-0 dann nur noch am Ende der Datei vorkommt. Gegenstück zum obigen Extrembeispiel:

```
10  PRINT CHR$(4) „OPEN TEST“ : PRINT CHR$(4) „WRITE TEST“
20  FOR X = 0 TO 1000
30  FOR Y = 1 TO 99 : PRINT CHR$(32); : NEXT Y : PRINT
40  NEXT X
50  PRINT CHR$(4) „CLOSE“
```

Ein weiterer Grund spricht für die „Vorformatierung“ von Random-Files. Nicht-vorformatierte und sozusagen „at random“ beschriebene Files sind auf der Diskette nicht in homogener Reihenfolge (in auf- oder absteigenden Spuren) abgespeichert mit der Folge, daß die Zugriffsgeschwindigkeit entsprechend sinkt. Hinzu kommt, daß DOS bei Random-Files nur eine „unintelligente Buchführung“ macht, die sogar die Anlage von Randon-Files und/oder Random-Records zuläßt, die später nie mehr vollständig beschrieben werden können! Ein Beispiel:

```
100  L = 256 · 4
110  E = 400
120  PRINT CHR$(4) „OPEN TEST,L“L
130  FOR R = 0 TO E
140  PRINT CHR$(4) „WRITE TEST,R“R
150  PRINT R + 1000
160  NEXT R
170  PRINT CHR$(4) „CLOSE“
```

(Die nachfolgenden Überlegungen setzen Kenntnisse aus dem Teil II dieses Buches voraus, so daß der Leser ggf. diesen Absatz überspringen möge.)

Bei diesem Programm beträgt die Feldlänge 4 Sektoren oder 256 mal 4 = 1024 Bytes = 1 Kilobyte. Es werden 401 (0-400) Records mit jeweils einer 4stelligen Zahl (1000-1400) + Return beschrieben. Die Datei wird in Zeile 170 „ordnungsmäßig" ohne jegliche Fehlermeldung geschlossen. Ist jedoch wirklich alles in Ordnung? Gewiß nicht! Denn eine normale DOS 3.3-Diskette mit 140 Kilobytes Bruttospeicherkapazität kann natürlich nicht 401 Kilobytes an Daten umfassen. Was ist passiert? Da mit Zeile 150 nur ein — inklusive Return — 5 Bytes langes Feld auf die Diskette geschrieben wurde, hat DOS für jeden Record nur 1 Sektor belegt, jedoch bereits 3 weitere Sektoren vorgemerkt. Dieses Vormerken geschieht dadurch, daß in der TSL für je 3 reservierte Sektoren Nullen eingetragen werden. Die erste TSL für die ersten 122 Daten-Sektoren sowie der erste eigentliche Daten-Sektor seien nachstehend auszugsweise gelistet:

Track $13, Sektor $0F: Erster TSL-Sektor

00	15	0F	00	00	00	00	00	T. $15, S. $0F nächste TSL
00	00	00	00	13	0E	T2	S2	T. $13, S. $0E erster Daten-Sektor
T3	S3	T4	S4	13	0D	T2	S2	T. $12, S. $0D zweiter Daten-Sektor
T3	S3	T4	S4	13	0C	T2	S2	usw.
T3	S3	T4	S4	13	0B	T2	S2	
T3	S3	T4	S4	13	0A	00	00	So sieht die TSL tatsächlich aus.
00	00	00	00	13	09	00	00	
00	00	00	00	13	08	00	00	usw.

Track $13, Sektor $0E: Erster Daten-Sektor

B1	B0	B0	B0	8D	00	00	00	1000 + Return
00	00	00	00	00	00	00	00	usw.

Der erste Daten-Sektor ist aus der ersten TSL in der zweiten Zeile des Hex-Dumps durch 13 0E (= Track $13, Sektor $0E) erkenntlich. Danach stehen auf der Diskette 6 Nullen, die hier als T2 S2, T3 S3, T4 S4 kenntlich gemacht wurden. Gemeint ist damit, daß mit den Nullen die zweiten/dritten/vierten Daten-Tracks/Sektoren vorläufig in der TSL reserviert, jedoch noch nicht in der VTOC als belegt markiert wurden. Der Random-File benötigt damit zunächst nur 401 reine Daten-Sektoren sowie zu-

sätzlich mehrere TSL-Sektoren, die jedoch insgesamt auf die Datendiskette passen. Wollte man nun jedoch die 401 bereits angelegten Records vollständig beschreiben, d.h. jeweils bis zum 1024. Byte, dann würde bereits nach weniger als 100 Records die Fehlermeldung „DISK FULL“ angezeigt.

Aus dem Beispiel kann man lernen, daß die Anlage dynamischer Random-Files, bei denen die Inhalte der Records und/oder die Anzahl der Records selbst kontinuierlich wächst, unter DOS nicht empfehlenswert ist.

2.6.9. APPEND, POSITION, BYTE

Diese DOS-Befehle werden selten benutzt, zumal insbesondere APPEND und POSITION zeitintensiv sind, d.h. unter Umständen beim normalen DOS 3.3 mehrere Minuten dauern können.

APPEND bewirkt das Anhängen (to append) eines speicherinternen Arrays an einen speicherexternen Textfile. Die Syntax ist APPEND — WRITE — PRINT. Ein vorangehendes OPEN ist überflüssig. Man beachte, daß APPEND definitionsgemäß nur für WRITE-Operationen gedacht. Beispiel:

```
10  PRINT CHR$(4) „OPEN TEST“
20  PRINT CHR$(4) „WRITE TEST“
30  FOR X = 0 TO 99 : PRINT X : NEXT
40  PRINT CHR$(4) „CLOSE“
50  REM JETZT FOLGT APPEND-BEFEHL
60  PRINT CHR$(4) „APPEND TEST“
70  PRINT CHR$(4) „WRITE TEST“
80  FOR X = 100 TO 199 : PRINT X : NEXT
90  PRINT CHR$(4) „CLOSE“
```

Die Felder eines Textfiles werden vom DOS bei bestimmten Befehlen wie numeriert behandelt, wobei die Numerierung mit Null beginnt (0, 1, 2, 3...). Symbolisch läßt sich dies so darstellen:

```
F0...rF1...rF2...rF3...rF4...rF5...rFn...Ctrl-0
           -Z-
           aP2
           rP0
```

d.h. Feld 0, gefolgt von Return, Feld 1, gefolgt von Return usw. bis zum Feld n, gefolgt von Return. Ein Feld besteht in der Regel aus 1...n Zeichen + Return, doch wird als Grenzfall vom DOS auch das nackte Return ohne vorangehende Zeichen als Feld aufgefaßt. Wenn ein File gerade geöffnet wurde, ist der Positionszeiger auf Feld 0 gerichtet. Nach dem ersten PRINT oder INPUT richtet sich der Zeiger auf Feld 1 usw. Jedes eingelesene oder gespeicherte Return erhöht also den Positionszeiger. Beim APPEND wird (bei DOS 3.3) der File solange eingelesen, bis kein Return mehr folgt, also der Endmarker Ctrl-0 erreicht wurde. Dann wird ab dieser Stelle der Array gespeichert. Bei diesem Verfahren ist es nicht verwunderlich, daß der APPEND-Befehl so lange dauert. (Diversi-DOS ist hier etwas „schlauer", da es gleich aufgrund der TSL zum letzten Datensektor geht, wodurch der APPEND-Befehl dann nur einen Bruchteil der Zeit in Anspruch nimmt.)

Man muß unterscheiden zwischen der relativen und der absoluten Feldposition. Wenn der Positionszeiger auf Feld 0 gerichtet ist, fallen absolute und relative Feldposition zusammen. Wenn der Zeiger (Z) jedoch z.B. nach 2 INPUTs von Feld 0 und 1 auf Feld 2 gerichtet wird, befindet man sich aus der Sicht von Feld 0 in der (absoluten) Position 2 (aP2) und aus der Sicht von Feld 2 in der relativen Position 0 (rP0); siehe obiges Schaubild.

Der R-Parameter bei Random-Files bezieht sich auf die absolute Position eines Records, der sich aus einem ODER mehreren, durch Returns getrennten Feldern zusammensetzen kann. Bei sequentiellen Files ist der R-Parameter ebenfalls möglich, doch bezieht er sich nunmehr auf die relative Feldposition, d.h. auf dasjenige Feld, daß dem r-ten Return vorausgeht. Nur direkt nach dem OPEN-Befehl ist die relative mit der absoluten Feldposition identisch. Beispiel:

```
10  PRINT CHR$(4) „OPEN TEST" : REM ZEIGER DANACH AUF F0
20  PRINT CHR$(4) „WRITE TEST"
30  PRINT 0 : REM ZEIGER DANACH AUF F1
40  PRINT 1 : REM ZEIGER DANACH AUF F2
50  PRINT 2 : REM ZEIGER DANACH AUF F3
60  PRINT 3 : REM ZEIGER DANACH AUF F4
70  PRINT 4 : REM ZEIGER DANACH AUF F5
80  PRINT CHR$(4) „CLOSE"
90  REM JETZT FOLGT EIGENTLICHER TEST
100 PRINT CHR$(4) „OPEN TEST" : REM ZEIGER DANACH AUF F0
110 PRINT CHR$(4) „POSITION TEST,R1" : REM RELATIV = ABSOLUT!
120 PRINT CHR$(4) „READ TEST"
130 INPUT R : PRINT R : REM FELD 1, WEIL 0 + 1 = 1!
```

```
140 PRINT CHR$(4) ,,POSITION TEST,R4" : REM RELATIV!
150 PRINT CHR$(4) ,,READ TEST"
160 INPUT R : PRINT R : REM FELD 5, WEIL 1 + 4 = 5!
170 PRINT CHR$(4) ,,CLOSE"
```

R0 ist das momentane Feld, R1 das nächste, R2 das übernächste usw. Aus der Sicht von z.B. dem absoluten Feld F1 ist R4 das (1 + 4 =) 5. Feld. Nach OPEN-POSITION-WRITE bzw. OPEN-POSITION-READ gilt: absolut gleich relativ. Später nach POSITION-WRITE bzw. POSITION-READ gilt: absolut ungleich relativ. Das nachfolgende Demo-Programm faßt die neuen Befehle zusammen:

```
100 REM = = = APPEND-POSITION-TEST = = =
105 REM --- MERGE.FILE DRIVE 1 ---
110 PRINT CHR$ (4) ,,OPEN MERGE.FILE,D1"
120 PRINT CHR$ (4) ,,DELETE MERGE.FILE"
130 PRINT CHR$ (4) ,,OPEN MERGE.FILE"
140 PRINT CHR$ (4) ,,WRITE MERGE.FILE"
150 Z = 0 : PRINT ,,00000" : REM FORMATIERTER FELDZÄHLER
160 PRINT CHR$ (4) ,,CLOSE MERGE.FILE"
165 REM --- EINZELFILE.1 DRIVE 2 ---
170 PRINT CHR$ (4) ,,OPEN EINZELFILE.1,D2"
180 PRINT CHR$ (4) ,,DELETE EINZELFILE.1"
190 PRINT CHR$ (4) ,,OPEN EINZELFILE.1"
200 PRINT CHR$ (4) ,,WRITE EINZELFILE.1"
210 FOR X = 0 TO 500
220 PRINT STR$ (X) + ,, AAAAAAAAAAAAAAAAAAAA"
230 NEXT X
240 PRINT CHR$ (4) ,,CLOSE EINZELFILE.1"
245 REM --- EINZELFILE.2 DRIVE 2 ---
250 PRINT CHR$ (4) ,,OPEN EINZELFILE.2,D2"
260 PRINT CHR$ (4) ,,DELETE EINZELFILE.2"
270 PRINT CHR$ (4) ,,OPEN EINZELFILE.2"
280 PRINT CHR$ (4) ,,WRITE EINZELFILE.2"
290 FOR X = 0 TO 500
300 PRINT STR$ (X) + ,, BBBBBBBBBBBBBBBBBBBB"
310 NEXT X
320 PRINT CHR$ (4) ,,CLOSE EINZELFILE.2"
330 HOME
340 PRINT ,,EINGABE M = RÜCKKEHR ZU MENÜ" : POKE 34, 2
350 REM ---------- MENÜ ----------
```

```
360  HOME :B$ = „1 = EINZELFILE.1 = 2 = EINZELFILE.2 "
370  PRINT „1 ABSOLUTE POSITION"
380  PRINT „2 RELATIVE POSITION"
390  PRINT „3 APPEND"
400  PRINT : INPUT „";K$
410  HOME: ON VAL (K$) GOTO 420, 550, 670 : GOTO 360
415  REM ----- ABSOLUT -----
420  INPUT „WELCHE ABSOLUTE POSITION ";A$
430  IF A$ = „M" GOTO 360
440  PRINT B$;: INPUT „";Y$
450  IF Y$ = „1" THEN Y$ = „EINZELFILE.1": GOTO 470
460  Y$ = „EINZELFILE-2"
470  A = VAL (A$)
480  PRINT CHR$ (4) „OPEN" Y$ „,D2" : ZEIGER AUF NULL
490  PRINT CHR$ (4) „POSITION" Y$ „,R"A
500  PRINT CHR$ (4) „READ" Y$
510  INPUT X$
520  PRINT CHR$ (4) „CLOSE" Y$
530  PRINT X$
540  GOTO 420
545  REM ----- RELATIV -----
550  INPUT „WELCHE RELATIVE POSITION? ";A$
560  IF A$ = „M" GOTO 360
570  A = VAL (A$)
580  PRINT B$;: INPUT „";Y$
590  IF Y$ = „1" THEN Y$ = „EINZELFILE.1": GOTO 610
600  Y$ = „EINZELFILE-2"
610  PRINT CHR$ (4) „POSITION" Y$ „,R"A : ZEIGER RELATIV!
620  PRINT CHR$ (4) „READ" Y$
630  INPUT X$
640  PRINT CHR$ (4)
650  PRINT X$
660  GOTO 550
665  REM ----- APPEND -----
670  PRINT CHR$ (4) „APPEND MERGE.FILE,D1"
680  PRINT CHR$ (4) „WRITE MERGE.FILE"
690  PRINT X$
700  PRINT CHR$ (4) „CLOSE MERGE.FILE"
710  PRINT CHR$ (4) „OPEN MERGE.FILE,D1"
720  PRINT CHR$ (4) „WRITE MERGE.FILE"
```

```
730  Z = Z + 1: PRINT LEFT$ ( „00000“, 5 — LEN ( STR$ ( Z ) ) ) ; Z
740  PRINT CHR$ (4) „CLOSE MERGE.FILE“
750  PRINT CHR$ (4) „CATALOG,D2“ : REM ERSETZT POKE -21912, 2
760  GOTO 360
```

Verfolgt man den obigen Programmfluß, so stellt man fest, daß die Zeile 610 auch dann durchlaufen werden kann, wenn einer der beiden Einzelfiles durch Zeile 480 nicht zuvor geöffnet wurde, d.h. der POSITION-Befehl ist auch ohne vorangehenden OPEN-Befehl zulässig, setzt jedoch dann den Zeiger auf Null, als ob OPEN vorangegangen wäre.

Der BYTE-Befehl spezifiziert bei sequentiellen Dateien die absolute bytemäßige Stelle in einem File. Nehmen wir an, ein Brief umfasse 2.001 Zeichen (0-2000). Dann würde das Programm

```
10  PRINT CHR$(4) „OPEN BRIEF“
20  PRINT CHR$(4) „READ BRIEF,B1000“
30  FOR X = 1000 TO 2000
40  GET X$ : PRINT CHR$(4); : PRINT X$; NEXT X
50  PRINT : PRINT CHR$(4) „CLOSE“
```

die zweite Hälfte des Briefes einlesen und am Bildschirm anzeigen. Im Gegensatz zu dem R-Parameter beim POSITION-Befehl, der sich immer nur auf NACHFOLGENDE Felder richten kann, kann der B-Parameter bei sequentiellen Textfiles ähnlich wie der R-Parameter bei Random-Files „vorwärts und rückwärts“ angewandt werden, d.h. mit dem BYTE-Befehl könnte man z.B. eine Datei rückwärts einlesen (was natürlich nicht besonders sinnvoll wäre):

```
10  PRINT CHR$(4) „OPEN ALPHABET“
20  PRINT CHR$(4) „WRITE ALPHABET“
30  FOR X = 65 TO 91 : PRINT CHR$(X); : NEXT : REM A BIS Z
40  PRINT : PRINT CHR$(4) „CLOSE“
50  PRINT CHR$(4) „OPEN ALPHABET“
60  FOR X = 25 TO 0 : REM Z BIS A
70  PRINT CHR$(4) „READ ALPHABET,B“,X
80  GET X$ : PRINT CHR$(4);: PRINT X$ : NEXT X
90  PRINT CHR$(4);: PRINT : PRINT CHR$(4) „CLOSE“
```

Man beachte, daß man bei beiden Programmen wegen des GET-Befehls, der das Return unterdrückt, eine permanente CHR$(4)-DOS-Umschaltung vornehmen muß.

Da der BYTE-Befehl nicht nach Feldern, sondern nach absoluten Diskettenspeicherstellen vorgeht und insofern Returns wie jedes andere Zeichen behandelt, ist die Anwendung dieses Befehls normalerweise nur bei Dateien mit exakt bekannter Struktur sinnvoll.

Abschließend sei erwähnt, daß der POSITION- bzw. BYTE-Befehl R- bzw. B-Parameter im Bereich 0-32767 zuläßt.

2.6.10. EXEC (Executive Textfile)

EXEC ist die Abkürzung für execution (Befehlsausführung) bzw. executive textfile (Befehlsdatei). Darunter ist ein sequentieller Textfile zu verstehen, der aus Direktbefehlen besteht, die man normalerweise manuell über die Tastatur eingeben würde, die jedoch durch Starten des Execfiles automatisch ausgeführt werden. Execfiles erstellt man am besten mit einem Textverarbeitungsprogramm, das Textfiles erzeugt, beispielsweise mit dem Applewriter IIe, oder ersatzweise mit einem Applesoft-Programm. Da bei aktiven Execfiles Tastatureingaben verboten sind — diese würden nämlich als Execbefehle interpretiert — und ferner nicht jeder DOS- oder Applesoft-Befehl Bestandteil eines Execfiles sein darf, ist die Nutzanwendung dieses File-Typs auf wenige Spezialfälle begrenzt. Vereinfachtes Beispiel:

```
10 PRINT CHR$(4) „OPEN EXECUTIVE“
20 PRINT CHR$(4) „WRITE EXECUTIVE“
30 PRINT „RUN PROGRAMM 1“
40 PRINT „RUN PROGRAMM 2“
50 PRINT CHR$(4) „CLOSE“
```

Danach würde der manuell über die Tastatur eingegebene Befehl

EXEC EXECUTIVE

zunächst das Programm 1 und — nach Beendigung desselben — automatisch das Programm 2 starten.

2.6.10.1. Bload-Finder

Dieser Execfile, der z.B. mit dem Applewriter auf einer Diskette unter dem Namen BLOADFIND gespeichert sein mag, bewirkt durch den Befehl EXEC BLOAD-

FIND die Ermittlung der Startadresse und Länge eines zuvor geBLOADeten Binärsfiles in dezimal und hexadezimal. Die diversen Pokes dienen der Hexadezimalumrechnung. Das „?" steht für PRINT.

POKE 47097, PEEK (43634) + PEEK (43616) − INT ((PEEK (43634) + PEEK (43616)) / 256) · 256: POKE 47098, PEEK (43635) + PEEK (43617) + ((PEEK (43634) + PEEK (43616)) > 255)
HOME : ?„BLOAD-FINDER − LETZTER BLOAD:"
?„DEZ.": ?„ANF: "; PEEK (43634) + 256 · PEEK (43635): ?„LEN: "; PEEK (43616) + 256 · PEEK (43617): ?„END: "; PEEK (47097) + 256 · PEEK (47098)
?„HEX.": ?„ANF: $";: POKE 70, PEEK (43634): POKE 71, PEEK (43635): POKE 58, 64: POKE 59, 249: CALL 65209: ?: ?„LEN: $";: POKE 70, PEEK (43616): POKE 71, PEEK (43617): CALL 65209: ?: ?„END: $";: POKE 70, PEEK (47097): POKE 71, PEEK (47098): CALL 65209
?: ?„ NAME EINFÜGEN": ?: ?„ BSAVE.........,A"; PEEK (43634) + 256 · PEEK (43635); „,L"; PEEK (43616) + 256 · PEEK (43617): HTAB 1: VTAB 17

2.6.10.2. List-Maker

Dieser Execfile − zu starten mit z.B. EXEC LIST.MAKER − speichert das sich gerade im RAM befindliche Applesoft-Programm unter dem Namen LIST auf der Diskette als Textfile (!), der z.B. dann mit dem Applewriter bearbeitet und später wieder mit EXEC LIST in ein normales (binäres) Applesoft-Programm zurückverwandelt werden kann. Die Pokes dienen u.a. der Tastatureingabe, da der GET-Befehl nicht möglich wäre.

POKE 768, 44: POKE 769, 16: POKE 770, 192: POKE 771, 173: POKE 772, 0:POKE 773, 192: POKE 774, 16: POKE 775, 251: POKE 776, 44: POKE 777, 16: POKE 778, 192: POKE 779, 96
63999 POKE 51, 128: POKE 33, 33: ?: ?CHR$(4) „OPENLIST": ?CHR$(4) „DELETELIST": ?CHR$(4) „OPENLIST": ?CHR$(4) „WRITELIST": LIST − 63998: ?CHR$(4) „CLOSELIST": TEXT: RETURN
HOME: INVERSE: ?„LIST.MAKER": NORMAL: ?: ?„INSERT DATA DISK AND PRESS RETURN": CALL 768: GOSUB 63999: DEL 63999, 63999

TEIL II: DOS für Assembler-Programmierer

In diesem Teil II werden technische Details der Datenspeicherung, diverse Tips und Tricks (auch für Applesoft-Programmierer) sowie insbesondere der Direktzugriff auf einzelne Diskettensektoren auf Maschinenebene behandelt. Da die Assemblerprogrammierung nicht jedermanns Sache ist, enthält dieser Teil eine Vielzahl ausführlich kommentierter Komplettprogramme als Quell-Code, so daß der Leser die Programme oder Programmteile problemlos in seine eigenen Programme integrieren kann. Als Assembler wurde der „Big Mac" bzw. „Merlin" verwendet. Auf seltene Pseudo-Opcodes und Macros wurde bewußt in den Assemblerprogrammen verzichtet, damit eine Abänderung des jeweiligen Source-Codes für andere Assembler (z.B. LISA, S-C-Assembler usw.) sich problemlos gestaltet.

6.2.10. File-Reader

```
100  REM *** FILE-READER ***
110  PRINT  CHR$ (21): IF  PEEK (36864) = 76 AND  PEEK (36865) =
   26 AND  PEEK (36866) = 144 THEN 140
120  PRINT : PRINT  CHR$ (4)"MAXFILES 3": CLEAR : HIMEM: 4096
130  PRINT : PRINT  CHR$ (4)"BLOAD FILE-READER.OBJ"
140  TEXT : HOME : INVERSE : PRINT "* FILE-READER *": PRINT "*
   U.STIEHL/83 *": NORMAL : PRINT : PRINT "START J/N ";
150  GET X$: ON X$ = "J" GOTO 160: ON X$ <  > "N" GOTO 150:
   PRINT : END
160  HOME : INVERSE : PRINT "DISKETTE EINLEGEN": NORMAL : PRINT
   : PRINT "W = WEITER ";
170  GET X$: IF X$ <  > "W" THEN 170
180  HOME : PRINT : PRINT  CHR$ (4)"CATALOG": INPUT "DATEI:";X$:
   IF X$ = "" GOTO 140
190  HOME : INVERSE : PRINT "ESC=ENDE * SPACE=STOPP": NORMAL :
   PRINT
200  PRINT  CHR$ (4)"UNLOCK";X$
210  PR# 0: IN# 0: CALL 36864: REM $9000
220  CALL 1002: PRINT : PRINT "1=ERNEUT * 0=ENDE ";
230  GET X$: ON X$ = "1" GOTO 140: IF X$ <  > "0" THEN 230
```

```
:ASM
                 ORG   36864        ;$9000
            *
9000: 4C 1A 90   JMP   INITIAL
            *
            *========FILE-READER===========
            *
            IND1     EQU   $CE         ;$CF
            IND2     EQU   $FE         ;$FF
            TEXT     EQU   $FB2F
            HOME     EQU   $FC58
            PRINT    EQU   $FDED
            HEXOUT   EQU   $FDDA
            *
            * MAX. 5 TSL = 5*122*2=1220=$4C4
            TOTALTSL EQU   $8B00       ;-$8FC4
            *
            *MACROPUFFER = 122 SEKTOREN
            *
            MACPUFF  EQU   $1000       ;-$BA00
            *
            * MAXFILES MUSS 3 SEIN
            *
            TSLPUFF  EQU   $9700       ;ANFANG
            TSLPTR   EQU   $9701       ;POINTER
            TSL      EQU   $970C       ;EIG.TSL
            RWTS     EQU   $3D9
            RWTSLOCO EQU   $3E3
```

```
                    DOSWARM  EQU  $3D0
                    *
                    *        IOB-BLOCK
                    *
9003: 01            IOB      HEX  01          ;IMMER
9004: 60            SLOT     HEX  60
9005: 01            DRIVE    HEX  01
9006: 00            VOLUME   HEX  00
9007: 00            TRACK    HEX  00
9008: 00            SECTOR   HEX  00
9009: 14            DCTLOW   DFB  #<DCT
900A: 90            DCTHIGH  DFB  #>DCT
900B: 00            BUFLOW   HEX  00
900C: 00            BUFHIGH  HEX  00
900D: 00                     HEX  00          ;UNUSED
900E: 00                     HEX  00          ;COUNT:0
900F: 01            COMMAND  HEX  01          ;READ
9010: 00            DOSERROR HEX  00
9011: 00            EFFVOL   HEX  00
9012: 60            VORSLOT  HEX  60
9013: 01            VORDRIVE HEX  01
                    *
                    *        DCT
                    *
9014: 00 01 EF      DCT      HEX  0001EFD8    ;IMMER
9017: D8
                    *
9018: 00            YSAVER   HEX  00
9019: 00            XSAVER   HEX  00
                    *
                    * ACTIVE SLOT + DRIVE ERMITTELN
                    *
901A: 20 E3 03      INITIAL  JSR  RWTSLOCO
901D: 84 CE                  STY  IND1        ;LOWIOB
901F: 85 CF                  STA  IND1+1      ;HIGHIOB
9021: A0 01                  LDY  #1
9023: B1 CE                  LDA  (IND1),Y
9025: 8D 04 90               STA  SLOT
9028: C8                     INY
9029: B1 CE                  LDA  (IND1),Y
902B: 8D 05 90               STA  DRIVE
902E: 2C 10 C0               BIT  $C010
                    *
                    * TOTAL-TSL INITIALISIEREN
                    *
9031: A9 00                  LDA  #<TOTALTSL
9033: 85 CE                  STA  IND1
9035: A9 BB                  LDA  #>TOTALTSL
9037: 85 CF                  STA  IND1+1
9039: A9 00                  LDA  #<TSLPUFF
903B: 8D 0B 90               STA  BUFLOW
903E: A9 97                  LDA  #>TSLPUFF
9040: 8D 0C 90               STA  BUFHIGH
9043: 4C 5D 90               JMP  SSTORE1     ;ERSTE
```

```
9046: AD 01 97  NEXTTSL LDA  TSLPTR
9049: F0 29             BEQ  SSTORE4    ;LETZTE
904B: 8D 07 90          STA  TRACK
904E: AD 02 97          LDA  TSLPTR+1
9051: 8D 08 90          STA  SECTOR
9054: A9 90             LDA  #>IOB
9056: A0 03             LDY  #<IOB
9058: 20 D9 03          JSR  RWTS
905B: B0 1A             BCS  ERROR1
905D: A2 00     SSTORE1 LDX  ##$00
905F: A0 00             LDY  ##$00
9061: BD 0C 97  SSTORE2 LDA  TSL,X
9064: 91 CE             STA  (IND1),Y
9066: E6 CE             INC  IND1
9068: D0 02             BNE  SSTORE3
906A: E6 CF             INC  IND1+1
906C: E8        SSTORE3 INX
906D: E0 F4             CPX  #244       ;122*2
906F: D0 F0             BNE  SSTORE2    ;MAX.
9071: 4C 46 90          JMP  NEXTTSL
9074: 4C E5 90  SSTORE4 JMP  BATCH0
                *
                * LESEFEHLER
                *
9077: 20 2F FB  ERROR1  JSR  TEXT
907A: 20 58 FC          JSR  HOME
907D: AD 10 90          LDA  DOSERROR
9080: 20 DA FD          JSR  HEXOUT
9083: 4C D0 03          JMP  DOSWARM
                *
                * LOESCHEN $1000-$8AFF
                *
9086: A9 00     CLEAR0  LDA  #<MACPUFF
9088: 85 FE             STA  IND2
908A: A9 10             LDA  #>MACPUFF
908C: 85 FF             STA  IND2+1
908E: A9 00             LDA  #0
9090: A8                TAY
9091: 91 FE     CLEAR1  STA  (IND2),Y
9093: C8                INY
9094: D0 FB             BNE  CLEAR1
9096: E6 FF             INC  IND2+1
9098: A6 FF             LDX  IND2+1
909A: E0 8B             CPX  #>TOTALTSL
909C: D0 F3             BNE  CLEAR1
909E: 60                RTS
                *
                * PRINT 122 SEKTOREN
                *
909F: A9 00     PRINT0  LDA  #0
90A1: 85 FE             STA  IND2
90A3: A9 10             LDA  #>MACPUFF
90A5: 85 FF             STA  IND2+1
90A7: A0 00             LDY  #0
90A9: B1 FE     PRINT1  LDA  (IND2),Y
```

```
90AB: 09 80     136          ORA  ##$80
90AD: C9 8D     137          CMP  ##$8D
90AF: F0 07     138          BEQ  PRINT2
90B1: C9 A0     139          CMP  ##$A0
90B3: B0 03     140          BCS  PRINT2
90B5: 38        141          SEC
90B6: E9 80     142          SBC  ##$80
90B8: 20 ED FD  143 PRINT2   JSR  PRINT
90BB: AD 00 C0  144          LDA  $C000
90BE: 10 13     145          BPL  PRINT3       ;NO KEY
90C0: 2C 10 C0  146          BIT  $C010
90C3: C9 9B     147          CMP  ##$9B        ;ESC
90C5: F0 1B     148          BEQ  PRINT4
90C7: C9 A0     149          CMP  ##$A0        ;SPACE
90C9: D0 08     150          BNE  PRINT3
90CB: AD 00 C0  151 WAIT     LDA  $C000
90CE: 10 FB     152          BPL  WAIT
90D0: 2C 10 C0  153          BIT  $C010
90D3: C8        154 PRINT3   INY
90D4: D0 D3     155          BNE  PRINT1
90D6: E6 FF     156          INC  IND2+1
90D8: A5 FF     157          LDA  IND2+1
90DA: CD 0C 90  158          CMP  BUFHIGH
90DD: 90 CA     159          BCC  PRINT1
90DF: F0 C8     160          BEQ  PRINT1
90E1: 60        161          RTS               ;RETURN
90E2: 68        162 PRINT4   PLA
90E3: 68        163          PLA
90E4: 60        164          RTS               ;ENDE
                165 *
                166 * 122 SEKTOREN BATCH LESEN
                167 *
90E5: A0 00     168 BATCH0   LDY  #0
90E7: 8C 18 90  169          STY  YSAVER
90EA: A9 00     170          LDA  #<TOTALTSL
90EC: 85 CE     171          STA  IND1
90EE: A9 8B     172          LDA  #>TOTALTSL
90F0: 85 CF     173          STA  IND1+1
                174 *
                175 * AUSSERER LOOP
                176 *
90F2: AC 18 90  177 BATCH1   LDY  YSAVER
90F5: B1 CE     178          LDA  (IND1),Y
90F7: D0 05     179          BNE  BATCH2
90F9: A9 00     180          LDA  #0
90FB: 85 48     181          STA  $48          ;P-REG
90FD: 60        182          RTS               ;ENDE
                183 *
                184 * READ A BATCH
                185 *
90FE: A2 00     186 BATCH2   LDX  #0
9100: 8D 19 90  187          STA  XSAVER
9103: A9 00     188          LDA  #<MACPUFF
9105: 8D 0B 90  189          STA  BUFLOW
9108: A9 10     190          LDA  #>MACPUFF
```

```
910A: 8D 0C 90            STA  BUFHIGH
910D: CE 0C 90            DEC  BUFHIGH
9110: 20 86 90            JSR  CLEAR0
                  *
                  * INNERER LOOP
                  *
9113: AC 18 90   BATCH3   LDY  YSAVER
9116: B1 CE               LDA  (IND1),Y
9118: F0 35               BEQ  BATCH7
911A: EE 0C 90            INC  BUFHIGH
911D: 8D 07 90            STA  TRACK
9120: EE 18 90            INC  YSAVER
9123: AC 18 90            LDY  YSAVER
9126: D0 02               BNE  BATCH4
9128: E6 CF               INC  IND1+1
912A: B1 CE      BATCH4   LDA  (IND1),Y
912C: 8D 08 90            STA  SECTOR
912F: A9 90               LDA  #>IOB
9131: A0 03               LDY  #<IOB
9133: 20 D9 03            JSR  RWTS
9136: 90 01               BCC  BATCH5
9138: EA                  NOP              ;ERROR1
9139: EE 18 90   BATCH5   INC  YSAVER
913C: AC 18 90            LDY  YSAVER
913F: D0 02               BNE  BATCH6
9141: E6 CF               INC  IND1+1
9143: AE 19 90   BATCH6   LDX  XSAVER
9146: E8                  INX
9147: E8                  INX
9148: 8E 19 90            STX  XSAVER
914B: E0 F4               CPX  #244
914D: 90 C4               BCC  BATCH3
914F: A9 00      BATCH7   LDA  #0
9151: 85 48               STA  $48         ;P-REG
9153: 20 9F 90            JSR  PRINT0
9156: 4C F2 90            JMP  BATCH1

--End assembly--

345 bytes

Errors: 0
```

6.2.11. FASTBRUN-Routine (Apple II Plus Emulator für IIe)

```
:ASM
                           ORG   $0D04
                   *
                   * FPBASIC FÜR APPLE IIE
                   * ---------------------
                   *
                   * FPBASIC WIRD IN DIE APPLE IIE
                   * LC-KARTE GELADEN. SO DASS DER
                   * APPLESOFT-INTERPRETER MODIFI-
                   * ZIERT WERDEN KANN.
                   *
                   * Dieses Programm ist ein Bei-
                   * spiel für die Anwendung der
                   * FASTBRUN-Routine.
                   * Wie wird's gemacht?
                   *
                   * 1. Dieses Driver-Programm
                   *    nach $0D04 BLOADen
                   * 2. FPBASIC von System-Master
                   *    nach $1000 BLOADen
                   * 3. BSAVE FP.NEU.A$0D04.L$32FB
                   * 4. Mit 'Bag of Tricks' oder
                   *    einer anderen RWTS-Utility
                   *    die ersten 4 Bytes des
                   *    ersten Sektors FP.NEU
                   *    von 04D0FB32
                   *    in   04D0F000 ändern.
                   *    so daß der BRUN-Befehl nur
                   *    noch den ersten Sektor
                   *    einliest und der Rest von
                   *    der Fastbrun-Routine besorgt
                   *    wird. (FP.NEU = 52 Sektoren)
                   *
                   *
                   * FASTBRUN-ROUTINE
                   * ================
                   *
                   * Diese Routine setzt MAXFILES 3
                   * und DOS 3.3 in den unteren 48K
                   * voraus!
                   *
0D04: 4C 1D 0D             JMP   PARAMS
                   IND1    EQU   $CE           ;$CE-CF
                   IND2    EQU   $FE           ;$FE-FF
                   TSL     EQU   $970C         ;MXFLS 3
                   RWTS    EQU   $3D9
                   *INPUT-OUTPUT-CONTROL-BLOCK*****
0D07: 01           IOB      HEX   01           ;STETS
0D08: 60           SLOT     HEX   60           ;SLOT 6
0D09: 01           DRIVE    HEX   01
0D0A: 00           VOLUME   HEX   00
```

```
0D0B: 00          TRACK     HEX  00
0D0C: 00          SECTOR    HEX  00
0D0D: 18          DCTLOW    DFB  #<DCT
0D0E: 0D          DCTHIGH   DFB  #>DCT
0D0F: 00          BUFLOW    HEX  00
0D10: 00          BUFHIGH   HEX  00             ;STETS 0
0D11: 00                    HEX  00             ;UNUSED
0D12: 00                    HEX  00             ;COUNT:0
0D13: 01          COMMAND   HEX  01             ;READ
0D14: 00          DOSERROR  HEX  00
0D15: 00          EFFVOL    HEX  00
0D16: 60          VORSLOT   HEX  60
0D17: 01          VORDRIVE  HEX  01
                  *DEVICE-CHARACTERISTICS-TABLE***
0D18: 00 01 EF    DCT       HEX  0001EFD8       ;STETS
0D1B: D8
0D1C: 00          TEMP      HEX  00
                  *
                  * SLOT/DRIVE/VOLUME
                  *
0D1D: 20 E3 03    PARAMS    JSR  $3E3           ;IOB WO?
0D20: 84 CE                 STY  IND1
0D22: 85 CF                 STA  IND1+1
0D24: A0 01                 LDY  #1
0D26: B1 CE                 LDA  (IND1),Y       ;IOB
0D28: 8D 08 0D              STA  SLOT
0D2B: 8D 16 0D              STA  VORSLOT
0D2E: C8                    INY
0D2F: B1 CE                 LDA  (IND1),Y
0D31: 8D 09 0D              STA  DRIVE
0D34: 8D 17 0D              STA  VORDRIVE
0D37: C8                    INY
0D38: B1 CE                 LDA  (IND1),Y
0D3A: 8D 15 0D              STA  EFFVOL
                  *
0D3D: A9 0D                 LDA  #$0D           ;0E00-1F
                  *
                  * Diese RWTS setzt das Einlesen
                  * mit dem 2. Sektor fort.
                  *
0D3F: 8D 10 0D              STA  BUFHIGH
0D42: A9 00                 LDA  #$00
0D44: 8D 0F 0D              STA  BUFLOW
0D47: A0 02                 LDY  #2
0D49: 8C 1C 0D              STY  TEMP           ;2.SEKT.
                  *
0D4C: EE 10 0D    READER1   INC  BUFHIGH
0D4F: AC 1C 0D              LDY  TEMP
0D52: B9 0C 97              LDA  TSL,Y
0D55: F0 26                 BEQ  READEND
0D57: 8D 0B 0D              STA  TRACK
0D5A: C8                    INY
0D5B: B9 0C 97              LDA  TSL,Y
0D5E: 8D 0C 0D              STA  SECTOR
```

```
0D61: EE 1C 0D  104               INC  TEMP
0D64: EE 1C 0D  105               INC  TEMP
0D67: A9 0D     106               LDA  #>IOB
0D69: A0 07     107               LDY  #<IOB
0D6B: 20 D9 03  108               JSR  RWTS
0D6E: B0 03     109               BCS  ERROR
0D70: 4C 4C 0D  110               JMP  READER1
0D73: A9 00     111   ERROR       LDA  #0
0D75: 85 48     112               STA  $48
0D77: A9 C5     113               LDA  #"E"
0D79: 8D 00 04  114               STA  $400
0D7C: 60        115               RTS
0D7D: A9 00     116   READEND     LDA  #0
0D7F: 85 48     117               STA  $48          ;P-REG
                118   *
                119   * Ende der Fastbrun-Routine
                120   *
                121   *
0D81: 4C 00 0E  122               JMP  $0E00
                123   *
                124   * AUF VOLLE PAGE AUFFÜLLEN!
                125   *
0D84: EA EA EA  126               HEX  EAEAEAEAEA
0D87: EA EA
0D89: EA EA EA  127               HEX  EAEAEAEAEA
0D8C: EA EA
0D8E: EA EA EA  128               HEX  EAEAEAEAEA
0D91: EA EA
0D93: EA EA EA  129               HEX  EAEAEAEAEA
0D96: EA EA
0D98: EA EA EA  130               HEX  EAEAEAEAEA
0D9B: EA EA
0D9D: EA EA EA  131               HEX  EAEAEAEAEA
0DA0: EA EA
0DA2: EA EA EA  132               HEX  EAEAEAEAEA
0DA5: EA EA
0DA7: EA EA EA  133               HEX  EAEAEAEAEA
0DAA: EA EA
0DAC: EA EA EA  134               HEX  EAEAEAEAEA
0DAF: EA EA
0DB1: EA EA EA  135               HEX  EAEAEAEAEA
0DB4: EA EA
0DB6: EA EA EA  136               HEX  EAEAEAEAEA
0DB9: EA EA
0DBB: EA EA EA  137               HEX  EAEAEAEAEA
0DBE: EA EA
0DC0: EA EA EA  138               HEX  EAEAEAEAEA
0DC3: EA EA
0DC5: EA EA EA  139               HEX  EAEAEAEAEA
0DC8: EA EA
0DCA: EA EA EA  140               HEX  EAEAEAEAEA
0DCD: EA EA
0DCF: EA EA EA  141               HEX  EAEAEAEAEA
0DD2: EA EA
```

```
0DD4: EA EA EA  142          HEX   EAEAEAEAEA
0DD7: EA EA
0DD9: EA EA EA  143          HEX   EAEAEAEAEA
0DDC: EA EA
0DDE: EA EA EA  144          HEX   EAEAEAEAEA
0DE1: EA EA
0DE3: EA EA EA  145          HEX   EAEAEAEAEA
0DE6: EA EA
0DE8: EA EA EA  146          HEX   EAEAEAEAEA
0DEB: EA EA
0DED: EA EA EA  147          HEX   EAEAEAEAEA
0DF0: EA EA
0DF2: EA EA EA  148          HEX   EAEAEAEAEA
0DF5: EA EA
0DF7: EA EA EA  149          HEX   EAEAEAEAEA
0DFA: EA EA
0DFC: EA EA EA  150          HEX   EAEAEAEA
0DFF: EA
                151  *
                152  *
                153  * FPBASIC + MONITOR APPLE II PLUS
                154  * FOR APPLE IIE WITH 64K CARD
                155  *
                156  FPBASIC1 EQU   $1000
                157  FPBASIC2 EQU   $D000
                158  DOSWARM  EQU   $03D0
                159  PRINT    EQU   $FDED
                160  HOME     EQU   $FC58
                161  *
                162  * EXIT, IF BRUN FROM 64K CARD
                163  *
0E00: AD 13 C0  164          LDA   $C013       ;AUXRAM?
0E03: 10 01     165          BPL   START       ;NO!
0E05: 60        166          RTS
                167  *
                168  * DISABLE 80-COL-CARD
                169  *
0E06: A9 95     170  START   LDA   #$95        ;CTRL-U
0E08: 20 ED FD  171          JSR   PRINT
0E0B: 20 58 FC  172          JSR   HOME
                173  *
                174  * SET SOFT-SWITCHES
                175  *
0E0E: 8E 0B C0  176          STX   $C00B       ;C3SLOT
0E11: 8E 00 C0  177          STX   $C000       ;80-OFF
0E14: 8E 0C C0  178          STX   $C00C       ;40-COL
0E17: 8E 0F C0  179          STX   $C00F       ;ALTCHR
0E1A: AE 83 C0  180          LDX   $C083       ;RD-LC
0E1D: AE 83 C0  181          LDX   $C083       ;WR-LC
                182  *
                183  * MOVE FP + MON ($1000-$3FFF)
                184  * TO LANGUAGE CARD BANK 2
                185  *
0E20: A9 00     186          LDA   #<FPBASIC1
```

```
0E22: 85 CE              STA  IND1
0E24: A9 10              LDA  #>FPBASIC1
0E26: 85 CF              STA  IND1+1
0E28: A9 00              LDA  #<FPBASIC2
0E2A: 85 FE              STA  IND2
0E2C: A9 D0              LDA  #>FPBASIC2
0E2E: 85 FF              STA  IND2+1
0E30: A0 00              LDY  #0
0E32: B1 CE       MOVE   LDA  (IND1),Y
0E34: 91 FE              STA  (IND2),Y
0E36: C8                 INY
0E37: D0 F9              BNE  MOVE
0E39: E6 CF              INC  IND1+1
0E3B: E6 FF              INC  IND2+1
0E3D: D0 F3              BNE  MOVE        ;<FFFF+1
                  *
                  * MONITOR PATCHES
                  *
0E3F: AD F5 0E           LDA  FD82
0E42: 8D 82 FD           STA  $FD82
0E45: AD F6 0E           LDA  FD82+1
0E48: 8D 83 FD           STA  $FD82+1
                  *
0E4B: AD F7 0E           LDA  FD11
0E4E: 8D 11 FD           STA  $FD11
0E51: AD F8 0E           LDA  FD11+1
0E54: 8D 12 FD           STA  $FD11+1
0E57: AD F9 0E           LDA  FD11+2
0E5A: 8D 13 FD           STA  $FD11+2
0E5D: AD FA 0E           LDA  FD11+3
0E60: 8D 14 FD           STA  $FD11+3
                  *
0E63: AD FB 0E           LDA  FBB3
0E66: 8D B3 FB           STA  $FBB3
                  *
0E69: AD FC 0E           LDA  FE80
0E6C: 8D 80 FE           STA  $FE80
0E6F: AD FD 0E           LDA  FE80+1
0E72: 8D 81 FE           STA  $FE80+1
                  *
                  * APPLESOFT PATCHES
                  *
0E75: AD FE 0E           LDA  F277
0E78: 8D 77 F2           STA  $F277
0E7B: AD FF 0E           LDA  F277+1
0E7E: 8D 78 F2           STA  $F277+1
                  *
0E81: AD 00 0F           LDA  F280
0E84: 8D 80 F2           STA  $F280
0E87: AD 01 0F           LDA  F280+1
0E8A: 8D 81 F2           STA  $F280+1
0E8D: AD 02 0F           LDA  F280+2
0E90: 8D 82 F2           STA  $F280+2
0E93: AD 03 0F           LDA  F280+3
```

```
0E96: 8D 83 F2               STA   $F280+3
0E99: AD 04 0F               LDA   F280+4
0E9C: 8D 84 F2               STA   $F280+4
                    *
0E9F: AD 05 0F               LDA   F181
0EA2: 8D 81 F1               STA   $F181
0EA5: AD 06 0F               LDA   F181+1
0EA8: 8D 82 F1               STA   $F181+1
0EAB: AD 07 0F               LDA   F181+2
0EAE: 8D 83 F1               STA   $F181+2
0EB1: AD 08 0F               LDA   F181+3
0EB4: 8D 84 F1               STA   $F181+3
0EB7: AD 09 0F               LDA   F181+4
0EBA: 8D 85 F1               STA   $F181+4
0EBD: AD 0A 0F               LDA   F181+5
0EC0: 8D 86 F1               STA   $F181+5
0EC3: AD 0B 0F               LDA   F181+6
0EC6: 8D 87 F1               STA   $F181+6
                    *
                    * RESET PATCHES
                    *
0EC9: AD F2 03               LDA   $03F2
0ECC: 8D 16 0F               STA   RESETOLD+1
0ECF: AD F3 03               LDA   $03F3
0ED2: 8D 17 0F               STA   RESETOLD+2
                    *
0ED5: A2 0B                  LDX   #11
0ED7: BD 0C 0F      RES      LDA   03C4,X
0EDA: 9D C4 03               STA   $03C4,X
0EDD: CA                     DEX
0EDE: 10 F7                  BPL   RES
                    *
0EE0: AD 18 0F               LDA   03F2
0EE3: 8D F2 03               STA   $03F2
0EE6: AD 19 0F               LDA   03F2+1
0EE9: 8D F3 03               STA   $03F2+1
0EEC: AD 1A 0F               LDA   03F2+2
0EEF: 8D F4 03               STA   $03F2+2
                    *
0EF2: 4C D0 03               JMP   DOSWARM
                    *
                    * MONITOR:
                    *
0EF5: 29 FF         FD82     AND   #$FF      ;CAPST
0EF7: 29 7F         FD11     AND   #$7F      ;RDKEY
0EF9: EA                     NOP
0EFA: EA                     NOP
0EFB: 06            FBB3     HEX   06        ;VERSION
0EFC: A0 7F         FE80     LDY   #$7F      ;SETINV
                    *
                    * APPLESOFT:
                    *
0EFE: A9 7F         F277     LDA   #$7F      ;INVERSE
0F00: A9 7F         F280     LDA   #$7F
```

```
0F02: 4C 79 F2                    JMP  $F279      ;FLASH
0F05: A9 C0         F181          LDA  #$C0
0F07: 85 51                       STA  $51
0F09: 4C 95 F1                    JMP  $F195      ;HIMEM
                    *
                    * RESET
                    *
                    RESETNEW EQU  $03C4      ;12BYTES
                    *
0F0C: 8D 0F C0      03C4          STA  $C00F      ;ALTCHR
0F0F: AD 83 C0                    LDA  $C083      ;RD-LC
0F12: AD 83 C0                    LDA  $C083      ;WR-LC
0F15: 4C BF 9D      RESETOLD JMP  $9DBF      ;POKED!
                    *
0F18: C4            03F2          DFB  #<RESETNEW
0F19: 03                          DFB  #>RESETNEW
0F1A: A6                          DFB  $03!$A5    ;PAGE 3

--End assembly--

535 bytes

Errors: 0
```

6502-Befehlstabelle

Function	Mnemon.	Accu.		Immedi.		Zero-P.		Zero,X		Zero,Y		Absol.		Abs,X		Abs,Y		(Ind,X)		(Ind),Y		Implied		Relative		Indirect		Status
		Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	N Z C I D V
Load	LDA			A9	2	A5	3	B5	4			AD	4	BD	4*	B9	4*	A1	6	B1	5*							N Z – – – –
	LDX			A2	2	A6	3			B6	4	AE	4			BE	4*											N Z – – – –
	LDY			A0	2	A4	3	B4	4			AC	4	BC	4*													N Z – – – –
Store	STA					85	3	95	4			8D	4	9D	5	99	5	81	6	91	6							– – – – – –
	STX					86	3			96	4	8E	4															– – – – – –
	STY					84	3	94	4			8C	4															– – – – – –
Transfer	TAX																					AA	2					N Z – – – –
	TXA																					8A	2					N Z – – – –
	TAY																					A8	2					N Z – – – –
	TYA																					98	2					N Z – – – –
Stack Ptr	TSX																					BA	2					N Z – – – –
	TXS																					9A	2					– – – – – –
Stack	PHA																					48	3					– – – – – –
	PLA																					68	4					N Z – – – –
Status R.	PHP																					08	3					– – – – – –
	PLP																					28	4					N Z C I D V
Flags	CLC																					18	2					– – 0 – – –
	SEC																					38	2					– – 1 – – –
	CLI																					58	2					– – – 0 – –
	SEI																					78	2					– – – 1 – –
	CLD																					D8	2					– – – – 0 –
	SED																					F8	2					– – – – 1 –
	CLV																					B8	2					– – – – – 0
Jump	JMP											4C	3													6C	5	– – – – – –
	JSR											20	6															– – – – – –
Return	RTS																					60	6					– – – – – –
	RTI																					40	6					N Z C I D V
Compare	CMP			C9	2	C5	3	D5	4			CD	4	DD	4*	D9	4*	C1	6	D1	5*							N Z C – – –
	CPX			E0	2	E4	3					EC	4															N Z C – – –
	CPY			C0	2	C4	3					CC	4															N Z C – – –
	BIT					24	3					2C	4															– Z – – – –
Branch	BMI																							30	2+			– – – – – –
N=0	BPL																							10	2+			– – – – – –
	BEQ																							F0	2+			– – – – – –
Z=0	BNE																							D0	2+			– – – – – –
	BCS																							B0	2+			– – – – – –
C=0	BCC																							90	2+			– – – – – –
	BVS																							70	2+			– – – – – –
V=0	BVC																							50	2+			– – – – – –
Increment	INC					E6	5	F6	6			EE	6	FE	7													N Z – – – –
	INX																					E8	2					N Z – – – –
	INY																					C8	2					N Z – – – –
Decrement	DEC					C6	5	D6	6			CE	6	DE	7													N Z – – – –
	DEX																					CA	2					N Z – – – –
	DEY																					88	2					N Z – – – –
Add/Subt.	ADC			69	2	65	3	75	4			6D	4	7D	4*	79	4*	61	6	71	5*							N Z C – – V
	SBC			E9	2	E5	3	F5	4			ED	4	FD	4*	F9	4*	E1	6	F1	5*							N Z C – – V
Boolean	AND			29	2	25	3	35	4			2D	4	3D	4*	39	4*	21	6	31	5*							N Z – – – –
	ORA			09	2	05	3	15	4			0D	4	1D	4*	19	4*	01	6	11	5*							N Z – – – –
	EOR			49	2	45	3	55	4			4D	4	5D	4*	59	4*	41	6	51	5*							N Z – – – –
Shift	ASL	0A	2			06	5	16	6			0E	6	1E	7													N Z C – – –
	LSR	4A	2			46	5	56	6			4E	6	5E	7													N Z C – – –
Rotate	ROL	2A	2			26	5	36	6			2E	6	3E	7													N Z C – – –
	ROR	6A	2			66	5	76	6			6E	6	7E	7													N Z C – – –
Miscell.	NOP																					EA	2					– – – – – –
	BRK																					00	7					– – – 1 – –
No. of Bytes		1		2		2		2		2		3		3		3		2		2		1		2		3		

* = 1 Takt mehr bei Seitenübergang;
\+ = 2 Takte ohne Verzweigung, 3 Takte bei Verzweigung, 4 Takte bei Verzweigung mit Seitenübergang.

ASCII-Tabelle

§	00	00000000	000	§	40	01000000	064	§	80	10000000	128	§	C0	11000000	192
A	01	00000001	001	A	41	01000001	065	A	81	10000001	129	A	C1	11000001	193
B	02	00000010	002	B	42	01000010	066	B	82	10000010	130	B	C2	11000010	194
C	03	00000011	003	C	43	01000011	067	C	83	10000011	131	C	C3	11000011	195
D	04	00000100	004	D	44	01000100	068	D	84	10000100	132	D	C4	11000100	196
E	05	00000101	005	E	45	01000101	069	E	85	10000101	133	E	C5	11000101	197
F	06	00000110	006	F	46	01000110	070	F	86	10000110	134	F	C6	11000110	198
G	07	00000111	007	G	47	01000111	071	G	87	10000111	135	G	C7	11000111	199
H	08	00001000	008	H	48	01001000	072	H	88	10001000	136	H	C8	11001000	200
I	09	00001001	009	I	49	01001001	073	I	89	10001001	137	I	C9	11001001	201
J	0A	00001010	010	J	4A	01001010	074	J	8A	10001010	138	J	CA	11001010	202
K	0B	00001011	011	K	4B	01001011	075	K	8B	10001011	139	K	CB	11001011	203
L	0C	00001100	012	L	4C	01001100	076	L	8C	10001100	140	L	CC	11001100	204
M	0D	00001101	013	M	4D	01001101	077	M	8D	10001101	141	M	CD	11001101	205
N	0E	00001110	014	N	4E	01001110	078	N	8E	10001110	142	N	CE	11001110	206
O	0F	00001111	015	O	4F	01001111	079	O	8F	10001111	143	O	CF	11001111	207
P	10	00010000	016	P	50	01010000	080	P	90	10010000	144	P	D0	11010000	208
Q	11	00010001	017	Q	51	01010001	081	Q	91	10010001	145	Q	D1	11010001	209
R	12	00010010	018	R	52	01010010	082	R	92	10010010	146	R	D2	11010010	210
S	13	00010011	019	S	53	01010011	083	S	93	10010011	147	S	D3	11010011	211
T	14	00010100	020	T	54	01010100	084	T	94	10010100	148	T	D4	11010100	212
U	15	00010101	021	U	55	01010101	085	U	95	10010101	149	U	D5	11010101	213
V	16	00010110	022	V	56	01010110	086	V	96	10010110	150	V	D6	11010110	214
W	17	00010111	023	W	57	01010111	087	W	97	10010111	151	W	D7	11010111	215
X	18	00011000	024	X	58	01011000	088	X	98	10011000	152	X	D8	11011000	216
Y	19	00011001	025	Y	59	01011001	089	Y	99	10011001	153	Y	D9	11011001	217
Z	1A	00011010	026	Z	5A	01011010	090	Z	9A	10011010	154	Z	DA	11011010	218
Ä	1B	00011011	027	Ä	5B	01011011	091	Ä	9B	10011011	155	Ä	DB	11011011	219
Ö	1C	00011100	028	Ö	5C	01011100	092	Ö	9C	10011100	156	Ö	DC	11011100	220
Ü	1D	00011101	029	Ü	5D	01011101	093	Ü	9D	10011101	157	Ü	DD	11011101	221
↑	1E	00011110	030	↑	5E	01011110	094	↑	9E	10011110	158	↑	DE	11011110	222
_	1F	00011111	031	_	5F	01011111	095	_	9F	10011111	159	_	DF	11011111	223
	20	00100000	032	`	60	01100000	096		A0	10100000	160	`	E0	11100000	224
!	21	00100001	033	a	61	01100001	097	!	A1	10100001	161	a	E1	11100001	225
"	22	00100010	034	b	62	01100010	098	"	A2	10100010	162	b	E2	11100010	226
#	23	00100011	035	c	63	01100011	099	#	A3	10100011	163	c	E3	11100011	227
$	24	00100100	036	d	64	01100100	100	$	A4	10100100	164	d	E4	11100100	228
%	25	00100101	037	e	65	01100101	101	%	A5	10100101	165	e	E5	11100101	229
&	26	00100110	038	f	66	01100110	102	&	A6	10100110	166	f	E6	11100110	230
'	27	00100111	039	g	67	01100111	103	'	A7	10100111	167	g	E7	11100111	231
(	28	00101000	040	h	68	01101000	104	(	A8	10101000	168	h	E8	11101000	232
)	29	00101001	041	i	69	01101001	105	)	A9	10101001	169	i	E9	11101001	233
*	2A	00101010	042	j	6A	01101010	106	*	AA	10101010	170	j	EA	11101010	234
+	2B	00101011	043	k	6B	01101011	107	+	AB	10101011	171	k	EB	11101011	235
,	2C	00101100	044	l	6C	01101100	108	,	AC	10101100	172	l	EC	11101100	236
-	2D	00101101	045	m	6D	01101101	109	-	AD	10101101	173	m	ED	11101101	237
.	2E	00101110	046	n	6E	01101110	110	.	AE	10101110	174	n	EE	11101110	238
/	2F	00101111	047	o	6F	01101111	111	/	AF	10101111	175	o	EF	11101111	239
0	30	00110000	048	p	70	01110000	112	0	B0	10110000	176	p	F0	11110000	240
1	31	00110001	049	q	71	01110001	113	1	B1	10110001	177	q	F1	11110001	241
2	32	00110010	050	r	72	01110010	114	2	B2	10110010	178	r	F2	11110010	242
3	33	00110011	051	s	73	01110011	115	3	B3	10110011	179	s	F3	11110011	243
4	34	00110100	052	t	74	01110100	116	4	B4	10110100	180	t	F4	11110100	244
5	35	00110101	053	u	75	01110101	117	5	B5	10110101	181	u	F5	11110101	245
6	36	00110110	054	v	76	01110110	118	6	B6	10110110	182	v	F6	11110110	246
7	37	00110111	055	w	77	01110111	119	7	B7	10110111	183	w	F7	11110111	247
8	38	00111000	056	x	78	01111000	120	8	B8	10111000	184	x	F8	11111000	248
9	39	00111001	057	y	79	01111001	121	9	B9	10111001	185	y	F9	11111001	249
:	3A	00111010	058	z	7A	01111010	122	:	BA	10111010	186	z	FA	11111010	250
;	3B	00111011	059	ä	7B	01111011	123	;	BB	10111011	187	ä	FB	11111011	251
<	3C	00111100	060	ö	7C	01111100	124	<	BC	10111100	188	ö	FC	11111100	252
=	3D	00111101	061	ü	7D	01111101	125	=	BD	10111101	189	ü	FD	11111101	253
>	3E	00111110	062	ß	7E	01111110	126	>	BE	10111110	190	ß	FE	11111110	254
?	3F	00111111	063		7F	01111111	127	?	BF	10111111	191		FF	11111111	255

Register

Fehler-Nummern

RWTS-Fehler-Codes

08 = INIT-Fehler
10 = schreibgeschützt
20 = falsche Volume-Nummer
40 = I/O-Fehler

DOS-Fehler-Codes

1 = LANGUAGE NOT AVAILABLE
2 = RANGE ERROR
3 = RANGE ERROR
4 = WRITE PROTECTED
5 = END OF DATA
6 = FILE NOT FOUND
7 = VOLUME MISMATCH
8 = I/O-ERROR
9 = DISK FULL
10 = FILE LOCKED
11 = SYNTAX ERROR
12 = NO BUFFERS AVAILABLE
13 = FILE TYPE MISMATCH
14 = PROGRAM TOO LARGE
15 = NOT DIRECT COMMAND

ProDOS-Editor 1.0

Applesoft-Editor
unter ProDOS-Betriebssystem
von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1024-X

Mit diesem neuen Editor – übrigens der bislang einzige deutsche ProDOS-Editor – wird dem *Applesoft-Programmierer* ein Werkzeug zur effektiven Programmierung unter dem Betriebssystem ProDOS gegeben, denn die früheren Editoren sind allesamt unter ProDOS nicht mehr lauffähig.
Unter anderen sind folgende Features implementiert worden:

- Zeilenorientierter Editor mit jedem erdenklichen Redigierkomfort (Insert, Delete, Tab, Restore, freie Cursorbewegung in allen vier Richtungen, Eingabe von Ctrl-Buchstaben in Applesoft-Zeilen usw.)
- Renumber (Zeilen-Umnumerierung)
- Xreference (sortierte Variablenliste)
- Suchen von Tokens, Strings und Variablen
- dezimale und hexadezimale Umrechnung
- Ausführung von Monitorbefehlen aus dem Editor heraus
- Listen des Applesoft-Programms in speicherinterner Form als Hex-Dump
- Suchen von Hex-Folgen, Adressen oder Speicherstellen im gesamten RAM-Bereich einschließlich der Language Card
- frei definierbare Tastatur-Macrobefehle
- Block-Dump (Einlesen und Speichern von ProDOS-Diskettenblocks)

Der Applesoft-Editor liegt in einem von ProDOS geschützten Bereich und läßt sich per Tastendruck vorübergehend abschalten und ebenso einfach wieder aktivieren.

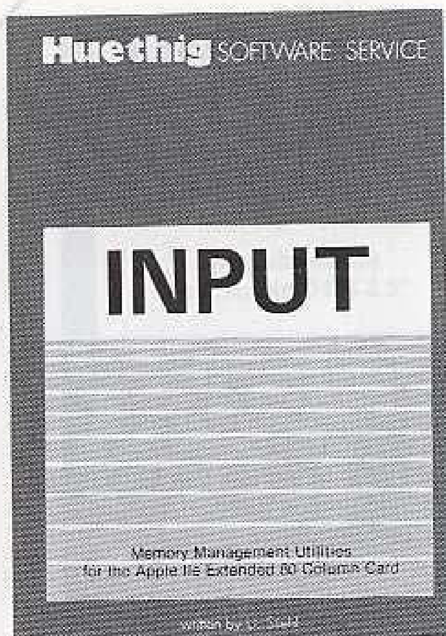


Begleitdiskette zu „Apple DOS 3.3 — Tips und Tricks“

DM 28,—

ISBN 3-7785-1032-0

A	002	STIEHL
T	004	BLOAD-FINDER
T	003	LIST-MAKER
B	004	TRACK-SECTOR-TRACER.S
T	004	T.TRACK-SECTOR-TRACER
B	002	TRACK-SECTOR-TRACER
B	003	FREIE SEKTOREN.S
T	002	T.FREIE SEKTOREN
B	002	FREIE SEKTOREN
B	006	FUSION-TASC.S
T	006	T.FUSION-TASC
B	002	FUSION-TASC
A	004	CPM-REFINER
B	006	CPM-REFINER.OBJ.S
T	006	T.CPM-REFINER.OBJ
B	002	CPM-REFINER.OBJ
B	008	DOS-OUTPUT-VEKTOR.S
T	008	T.DOS-OUTPUT-VEKTOR
B	002	DOS-OUTPUT-VEKTOR
B	011	RWTS-MUSTER.S
T	011	T.RWTS-MUSTER
B	002	RWTS-MUSTER
B	009	RWTS-TEST.WARTESCHLEIFE.S
T	009	T.RWTS-TEST.WARTESCHLEIFE
B	002	RWTS-TEST.WARTESCHLEIFE
B	005	RWTS-TEST.SKEWING.S
T	005	T.RWTS-TEST.SKEWING
B	002	RWTS-TEST.SKEWING
A	004	DISKLESER
B	007	DISKLESER.OBJ.S
T	007	T.DISKLESER.OBJ
B	002	DISKLESER.OBJ
B	018	KOPIERPROGRAMM.2DRIVE.S
T	018	T.KOPIERPROGRAMM.2DRIVE
B	004	KOPIERPROGRAMM.2DRIVE
B	020	KOPIERPROGRAMM.1DRIVE.S
T	020	T.KOPIERPROGRAMM.1DRIVE
B	005	KOPIERPROGRAMM.1DRIVE
B	008	DOS-KOPIE-TRACK 00-02.S
T	008	T.DOS-KOPIE-TRACK 00-02
B	002	DOS-KOPIE-TRACK 00-02
B	017	DISK-COMPARER.S
T	017	T.DISK-COMPARER
B	004	DISK-COMPARER
B	010	BAD-SECTOR-ROUTINE.S
T	010	T.BAD-SECTOR-ROUTINE
B	003	BAD-SECTOR-ROUTINE
B	017	DOS-LOSE DATENDISKETTE.S
T	017	T.DOS-LOSE DATENDISKETTE
B	004	DOS-LOSE DATENDISKETTE
B	039	DOSMOVER-RAMDISK-DRIVER.S
T	039	T.DOSMOVER-RAMDISK-DRIVER
B	006	DOSMOVER-RAMDISK-DRIVER
A	003	TSL-MAKER
B	007	TSL-MAKER.OBJ.S
T	007	T.TSL-MAKER.OBJ
B	002	TSL-MAKER.OBJ
A	004	FILE-READER
B	013	FILE-READER.OBJ.S
T	013	T.FILE-READER.OBJ
B	003	FILE-READER.OBJ
B	018	FASTBRUN-ROUTINE.S
T	018	T.FASTBRUN-ROUTINE



INPUT 2.0
Ein Bildschirm-Maskengenerator für DOS 3.3 und ProDOS

von U. Stiehl

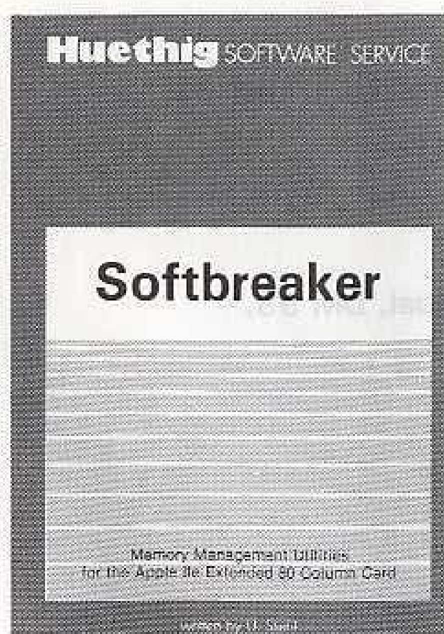
1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1021-5

Der für den Apple II bestimmte Maskengenerator „Input 2.0" basiert auf den früheren Programmen „Input 1.0" und „Input80 1.0" (von denen noch Restbestände lieferbar sind) und ist sowohl unter DOS 3.3 wie auch unter dem neuen ProDOS lauffähig. Der Maskengenerator setzt einen Apple II Plus mit Language Card oder einen Apple IIe voraus. Im 40 Z/Z-Modus funktioniert er auf beiden Gerätetypen, im 80 Z/Z-Modus dagegen nur auf dem Apple IIe mit 80-Zeichen-Karte. (Die alte Videx-Karte für den Apple II wird nicht unterstützt!)

„Input 2.0" liegt wahlweise in der Bank 1 oder Bank 2 der Language Card und wird durch einen kurzen Driver in den unteren 48K aufgerufen. „Input 2.0" läßt sich problemlos in nicht-compilierte und compilierte Applesoft- sowie in Assemblerprogramme einbinden. Die Übergabe der Feldinhalte an das Anwenderprogramm erfolgt durch ein einfaches Verfahren, das auch bei Compilern funktioniert.

Für jedes Feld der Bildschirmmaske lassen sich u. a. definieren: Feldlänge (bis zu 255 Zeichen) – Vtab – Htab – Datentyp (insgesamt 8 Typen) – Scrollflag (starre oder dynamische Maske) – Ctrlflag – Füllflag – Löschflag – Bildschirmflag (40- oder 80-Z-Darstellung). Innerhalb eines Eingabefeldes besteht jeder denkbare Redigierkomfort (Insert, Delete, Rubout, Restore usw.).

Bei der neuen Version des Maskengenerators können jetzt auch Ctrl-Zeichen beim Datentyp String eingegeben werden. Ferner sind – dies gilt nur für IIe – die Apfeltasten als schnelle Cursortasten definiert. Schließlich wurden Features implementiert, die den Einsatz von „Input 2.0" als zeilenorientiertes Textverarbeitungsprogramm ermöglichen. Die „Input 2.0"-Diskette enthält zahlreiche Demos zur Veranschaulichung der Anwendung.



Softbreaker 1.0

Eine softwaremäßige Interrupt-Utility für die Apple IIe 64K-Karte

von U. Stiehl

1984, Diskette und Manual, DM 48,–

ISBN 3-7785-1022-3

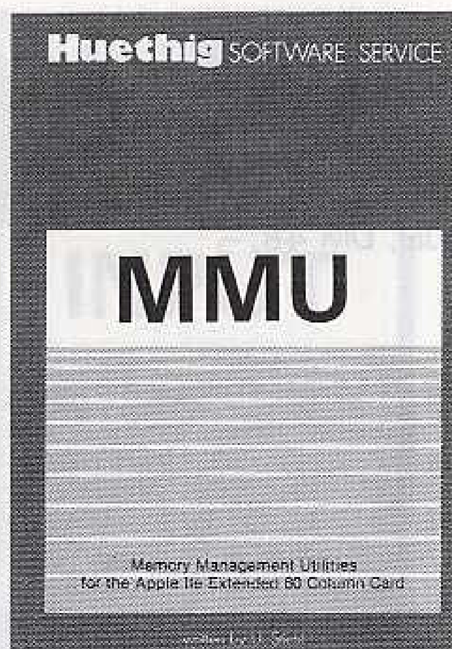
„Softbreaker" ist ein Assemblerprogramm, mit dessen Hilfe Programme, die sich von der 64K-Karte (= Extended 80 Column Card für den Apple IIe) starten lassen, unterbrochen, gespeichert, geladen und exakt an der Stelle der Unterbrechung fortgeführt werden können. Dadurch ist es auch möglich, Sicherungskopien von sogenannten kopiergeschützten Programmen herzustellen. Es sei hier ausdrücklich darauf hingewiesen, daß laut Urheberrechtsgesetz die Herstellung von Vervielfältigungsstücken nur von rechtmäßig erworbenen Werken und nur zum persönlichen Gebrauch zulässig ist.

„Softbreaker" läßt sich bei folgenden Programmtypen anwenden:

– bei grundsätzlich allen für den Apple II Plus konzipierten Programmen, die die 40-Zeichen-Darstellung und/oder die Grafikseite 1 (also nicht HGR2) benutzen, gleichviel ob sie durch Reset abgesichert sind oder nicht.
– bei grundsätzlich allen durch Reset abgesicherten Programmen, auch wenn sie die 80-Zeichen-Darstellung benutzen.

Softbreaker funktioniert selbstverständlich nicht bei Programmen, die die 64K-Karte des Apple IIe in irgendeiner Form benutzen, z. B. nicht bei Programmen, die die 64K-Karte als RAM-Disk oder als Zusatzspeicher verwenden.

Mit Softbreaker unterbrochene Programme werden komplett, d. h. die ganzen 64K einschließlich Language Card, in nur ca. 11 Sekunden auf einer formatierten Diskette gesichert, wobei die Rücksprungadresse und diverse Status-Werte automatisch mit abgespeichert werden, so daß keinerlei technischen oder Programmierkenntnisse erforderlich sind.



MMU 2.0
Memory Managements Utilities

für die Apple IIe 64K-Karte

DOS 3.3 (und ProDOS)

von U. Stiehl

1984, Diskette und Manual, DM 98,–

ISBN 3-7787-1023-1

Bei der Version 2.0 der MMU's sind die Utilities teilweise so umgeschrieben worden, daß sie sowohl unter DOS 3.3 als auch unter ProDOS lauffähig sind. Da dies nicht immer möglich war, sind zusätzlich zu den reinen DOS-Hilfsprogrammen, speziell den RAM-Disk-Drivern, einige reine ProDOS-Utilities aufgenommen worden. Insgesamt enthält die neue „MMU 2.0"-Diskette über 25 Programme, die neue Einsatzmöglichkeiten für die Extended 80 Column Card (erweiterte 80-Z-Karte = 64K-Karte für den Apple IIe) erschließen. Ein Teil der Programme laufen auch auf dem Apple II Plus, doch ist „MMU 2.0" primär für 64K-Karte-Besitzer gedacht.

Im einzelnen umfaßt „MMU 2.0"

- Drei RAM-Disk-Driver für DOS 3.3: „INIT62" benutzt nur die 64K-Karte als RAM-Disk, „INIT78" benutzt zusätzlich die Motherboard-LC als RAM-Karte und „DOSMOVER. INIT62" gilt für den Fall, daß sich das DOS selbst in der Motherboard-LC befindet.
- Eine sehr nützliche Pseudo-Coprocessor-Utility, die das Hin- und Herschalten zwischen zwei Programm-Modulen ermöglicht, von denen sich das eine Modul auf der 64K-Karte befindet.
- Zwei schnelle Kopierprogramme (für DOS 3.3 und ProDOS).
- Mehrere Move-Programme zum Verschieben von Daten auf die 64K-Karte sowie auf die Language Card und umgekehrt.
- Mehrere Hilfsprogramme zum Untersuchen und Löschen bestimmter Speicherbereiche der 64K-Karte und der LC, zur Ermittlung des Softswitch-Status usw.
- Zwei Simulator-Programme zum Simulieren von Apple II und Apple II Plus auf dem Apple IIe.

Schließlich enthält „MMU 2.0" auch verschiedene Move-Utilities für die RAM-Karten AP20 und AP17 der Firma IBS.

Dies ist die erste deutschsprachige Darstellung des Diskettenbetriebssystems DOS 3.3 für den Apple II/II Plus/IIe, die sich sowohl an Applesoft- als auch an Assembler-Programmierer wendet. Sinngemäß ist das Buch zweigeteilt:

Der erste Teil behandelt ausführlich die dem Applesoft-Programmierer zur Verfügung stehenden DOS-Befehle, wobei die Textfiles wegen ihrer großen Bedeutung und der vergleichsweise komplizierten Handhabung besonders dargestellt werden. Viele Textfile-Tricks aus der langjährigen Programmiererfahrung des Autors werden hier zum erstenmal geschildert.

Aber auch im zweiten Teil findet der reine Applesoft-Programmierer insbesondere in dem Kapitel „Vermischte Tips, Tricks und Patches" zahlreiche Anregungen. Im übrigen ist der zweite Teil für Assembler-Programmierer gedacht. Neben einer detaillierten Beschreibung der DOS-Interna enthält dieser Teil elf vollständige RWTS-Anwendungsprogramme — z.B. CPM-Refiner, DOS-lose Datendisk, TSL-Maker, File-Reader, Pseudo-Disk-Driver und Fastbrun-Routine —, die Techniken enthüllen, die bislang noch niemals publiziert worden sind. Dieses DOS-Buch ist deshalb der unentbehrliche Begleiter für jeden Apple-Programmierer.

